Teresa M. Corbin (SBN 132360)
Christopher Kelley (SBN 166608)
Erik K. Moller (SBN 147674)
HOWREY SIMON ARNOLD & WHITE, LLP
301 Ravenswood Avenue
Menlo Park, California 94025
Telephone: (650) 463-8100
Facsimile: (650) 463-8400

Attorneys for Plaintiff Synopsys, Inc.

UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

SYNOPSYS, INC.,

Plaintiff,

vs.

RICOH COMPANY, LTD., a Japanese corporation.,

Defendant.

Case No. C03 02289 PVT

COMPLAINT FOR DECLARATORY JUDGMENT

DEMAND FOR JURY TRIAL

Plaintiff Synopsys, Inc. ("Synopsys"), for its complaint against Ricoh Company, Ltd. ("Ricoh"), alleges:

PARTIES

1. Synopsys is a Delaware corporation having its principal place of business at 700 E. Middlefield Road, Mountain View, California.

2. Ricoh is, upon information and belief, a Japanese corporation having its principal place of business at 3-6 Nakamagome 1-chome, Ohta-ku Tokyo, Japan.

OVERVIEW

3. This action is brought to resolve the apprehension under which Synopsys is forced to conduct its business in the United States as a result of the threatened assertion of infringement of certain patents assigned to Ricoh.

4. Synopsys is a leading supplier of a type of Electronic Design Automation ("EDA") software known as logic synthesis software. One of Synopsys' most popular products is its "Design Compiler" synthesis tool.

5. Ricoh contends that it is the assignee and beneficial owner of United States patents U.S. Patent No. 4,922,432 (the "'432 Patent") and U.S. Patent No. 5,197,016 (the "'016 Patent"). These patents purport to disclose and claim a particular logic synthesis software tool. The '016 Patent issued as the result of a continuation in part of the same patent application that issued as the '432 Patent. The '432 Patent and the '016 Patent are attached to this complaint as Exhibits A and B respectively.

6. On January 21, 2003, Ricoh filed a patent infringement action against Aeroflex Inc. ("Aeroflex"), AMI Semiconductor, Inc. ("AMI"), and Matrox Electronic Systems, Ltd., Matrox Graphics Inc., Matrox International Corp., and Matrox Tech, Inc. (collectively "Matrox") in the United States District Court for the District of Delaware. Ricoh's suit alleges that each of the defendants infringe one or more of the claims of the '432 Patent. Aeroflex, AMI and some of the Matrox entities are users of Synopsys' software. Ricoh has indicated in communications to one of the defendants in the Delaware litigation that the basis of its infringement allegations is the defendants' use of Synopsys' Design Compiler product.

7. After filing the Delaware suit, Ricoh, through its counsel, sent letters to one or more users of Synopsys' software. Ricoh's letters assert that the recipients are using "a computer-aided design system obtained from Synopsys, including Design Compiler." Ricoh's letter then states that the '432 Patent and '016 Patent are "basic patents directed to computer-aided design processes," and that Ricoh is currently enforcing these patents in a lawsuit filed in the U.S. District Court in Delaware. The letter concludes with a request that the recipient contact Ricoh regarding licensing of the patents.

8. Based on these letters and the Delaware suit, Synopsys is informed and believes that Ricoh's purpose is to assert the '432 Patent and the '016 Patent against Synopsys' customers in an effort to extract licensing royalties from the use of Synopsys' products, including Design Compiler.

9. The original assignees to the '432 Patent and '016 Patent were International Chip Corporation ("ICC") and Ricoh. In 1991, ICC assigned its interests in both patents to Knowledge

Based Silicon Corporation ("KBS"). KBS subsequently assigned its interests in both patents to Ricoh in 2001.

10. Persons representing KBS met with representatives from Synopsys during the 1991 meeting of the Design Automation Conference, held in San Francisco. During that meeting, Synopsys was told that its Design Compiler product practiced certain claims of the '432 Patent. KBS' efforts to persuade Synopsys to obtain a license to the '432 Patent and the '016 Patent were unsuccessful and subsequently abandoned.

11. KBS developed and marketed products, including products named flowHDL and blockHDL, which were designed to be interoperable with Synopsys' Design Compiler product. In 1993 and 1995, in order to assist KBS with efforts to make its products interoperable with Synopsys' products, KBS and Synopsys entered into several agreements, including a license to Design Compiler, under which Synopsys disclosed confidential information about its products, and rendered other technical assistance, to KBS. At no point during these cooperative efforts did KBS make any allegations that Synopsys' products were infringing either the '432 Patent or the '016 Patent. Ricoh has been a licensed user of Synopsys' Design Compiler software since at least 1997.

JURISDICTION

12. This is an action for the resolution of an existing conflict under the Declaratory Judgment Act, 28 U.S.C. §§ 2201 and 2202. The underlying causes of action arise under the patent laws of the United States, including 35 U.S.C. §§ 271 and 282. Synopsys seeks a declaration by this Court that the '432 Patent and the '016 Patent are invalid, unenforceable, and not infringed by Synopsys and its customers. The subject matter jurisdiction of this Court is therefore established by 28 U.S.C. § 1338

13. The subject matter jurisdiction of this Court for all counts stated herein is also established by the general jurisdiction of district courts under United States law, 28 U.S.C. §§ 1331 and 1332.

14. Upon information and belief, this Court has personal jurisdiction over Ricoh because Ricoh transacts business, performs some character of work or service in California, or contracts to supply services or things in California and has sufficient minimum contacts with this state.

15. This action presents an actual case or controversy because the acts of Ricoh have created in Synopsys a reasonable apprehension that, as a result of Synopsys' continuing manufacture, use, and sale of its Design Compiler product in the United States, Synopsys will be sued for infringement of the '432 Patent and the '016 Patent. In addition, Synopsys fears that it will suffer commercial injury if Ricoh continues to sue, and threaten to sue, Synopsys' customers, as those customers will then be dissuaded from purchasing and using Design Compiler and other Synopsys logic synthesis products.

VENUE

16. Venue for this action is proper in this judicial district under 38 U.S.C. § 1391 (b) and (c).

COUNT I

Declaratory Judgment That U.S. Patent No. 4,922,432 Is Not Infringed By Any Synopsys Logic Synthesis Product

17. Synopsys repeats and realleges paragraphs 1 through 16 of this Complaint as if the same were fully set forth herein.

18. Synopsys has not made, used, offered to sell or sold, within the United States, or imported into the United States, any products or processes that infringe any valid claim of the '432 Patent, either directly, indirectly, contributorily or otherwise, and has not induced others to infringe the '432 Patent.

COUNT II

Declaratory Judgment That The Claims of U.S. Patent 4,922,432 Are Invalid

19. Synopsys repeats and realleges paragraphs 1 through 18 of this Complaint as if the same were fully set forth herein.

20. The claims of the '432 Patent are invalid for failure to meet the requirements specified in Title 35 of the United States Code, including, but not limited to, 35 U.S.C. §§ 101, 102, 103, and 112 including one or more of the following reasons: (a) the inventor named in the '432 Patent did not invent or discover any new useful process, machine, manufacture, or composition of matter, or any

new and useful improvement thereof within the meaning of 35 U.S.C. § 101; (b) the subject matter claimed in the '432 Patent was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before it was invented by the inventors named in the '432 Patent, as prohibited by 35 U.S.C. § 102(a); (c) the subject matter claimed in the '432 Patent was patented or described in a printed publication in this or a foreign country or was in public use or on sale in this country, more than one year prior to the filing of the application which resulted in the '432 Patent in the United States, as prohibited by 35 U.S.C. § 102(b); (d) the subject matter claimed in the '432 Patent was described in a United States patent based on an application filed in the United States or described in an application published prior to its invention by the inventors named in the '432 Patent, as prohibited by 35 U.S.C. § 102(e); (e) the inventor named in the '432 Patent did not invent the subject matter; (f) the subject matter claimed in the '432 Patent was invented in this country by another inventor, who did not abandon, suppress or conceal it, before its invention by the inventors named in the '432 Patent, as prohibited by 35 U.S.C. § 102(g); (g) the subject matter claimed in the '432 Patent would have been obvious, in view of the prior art, to a person having ordinary skill in the art at the time the invention was made under 35 U.S.C. § 103; and/or (h) the claims of the '432 Patent are invalid for failing to comply with 35 U.S.C. § 112, in that (i) the specification fails to contain a written description of the subject matter claimed in the '432 Patent and the manner and process of making and using it; (ii) the claims fail to particularly point out and distinctly claim a patentable invention, (iii) the claims are indefinite, (iv) the specification fails to enable one skilled in the art to practice the claimed invention, and/or (v) the specification fails to set forth the best mode contemplated by the named inventors for carrying out the alleged invention. Plaintiff reserves the right to amend this count further, as additional information is developed through discovery or otherwise.

COUNT III

Declaratory Judgment That U.S. Patent No. 5,197,016 Is Not Infringed By Any Synopsys Logic Synthesis Product

21. Synopsys repeats and realleges paragraphs 1 through 20 of this Complaint as if the same were fully set forth herein.

22. Synopsys has not made, used, offered to sell or sold, within the United States, or imported into the United States, any products or processes that infringe any valid claim of the '016 Patent, either directly, indirectly, contributorily or otherwise, and has not induced others to infringe the '016 Patent.

COUNT IV

Declaratory Judgment That The Claims of U.S. Patent 5,197,016 Are Invalid

23. Synopsys repeats and realleges paragraphs 1 through 22 of this Complaint as if the same were fully set forth herein.

24. The claims of the '016 Patent are invalid for failure to meet the requirements specified in Title 35 of the United States Code, including, but not limited to, 35 U.S.C. §§ 101, 102, 103, and 112 including one or more of the following reasons: (a) the inventor named in the '016 Patent did not invent or discover any new useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof within the meaning of 35 U.S.C. § 101; (b) the subject matter claimed in the '016 Patent was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before it was invented by the inventors named in the '016 Patent, as prohibited by 35 U.S.C. § 102(a); (c) the subject matter claimed in the '016 Patent was patented or described in a printed publication in this or a foreign country or was in public use or on sale in this country, more than one year prior to the filing of the application which resulted in the '016 Patent in the United States, as prohibited by 35 U.S.C. § 102(b); (d) the subject matter claimed in the '016 Patent was described in a United States patent based on an application filed in the United States or described in an application published prior to its invention by the inventors named in the '016 Patent, as prohibited by 35 U.S.C. § 102(e); (e) the inventor named in the '016 Patent did not invent the subject matter; (f) the subject matter claimed in the '016 Patent was invented in this country by another inventor, who did not abandon, suppress or conceal it, before its invention by the inventors named in the '016 Patent, as prohibited by 35 U.S.C. § 102(g); (g) the subject matter claimed in the '016 Patent would have been obvious, in view of the prior art, to a person having ordinary skill in the art at the time the invention was made under 35 U.S.C. § 103; and/or (h) the claims of the '016 Patent

are invalid for failing to comply with 35 U.S.C. § 112, in that (i) the specification fails to contain a written description of the subject matter claimed in the '016 Patent and the manner and process of making and using it; (ii) the claims fail to particularly point out and distinctly claim a patentable invention, (iii) the claims are indefinite, (iv) the specification fails to enable one skilled in the art to practice the claimed invention, and/or (v) the specification fails to set forth the best mode contemplated by the named inventors for carrying out the alleged invention. Plaintiff reserves the right to amend this count further, as additional information is developed through discovery or otherwise.

COUNT V

Declaratory Judgment Barring Ricoh From Recovery Under the Doctrine of Laches

25. Synopsys repeats and realleges paragraphs 1 through 24 of this Complaint as if the same were fully set forth herein.

26. Ricoh uses Synopsys' logic synthesis software and is familiar with the operation of this software.

27. Ricoh is barred under the equitable doctrine of laches from recovering damages for any past practice of its patents. Ricoh unreasonably and inexcusably delayed filing suit to the material prejudice of Synopsys and Synopsys' customers.

COUNT VI

Declaratory Judgment Barring Ricoh From Recovery Under the Doctrine of Equitable Estoppel

28. Synopsys repeats and realleges paragraphs 1 through 27 of this Complaint as if the same were fully set forth herein.

29. Ricoh is barred under the doctrine of equitable estoppel from seeking enforcement of its patent rights in the '432 Patent and '016 Patent. The conduct of the assignees of the '432 Patent and '016 Patent led Synopsys to believe that the assignees did not believe that those patents were infringed by any Synopsys product. Synopsys reasonably relied on assignees' conduct. In the light of Ricoh's attempts to enforce the '432 and '016 patents against Synopsys' products, Synopsys' reliance was detrimental to its interests.

RESERVATION OF CLAIMS

30. Synopsys reserves the right to assert any other claims that discovery may reveal, including, but not limited to, inequitable conduct.

PRAYER FOR RELIEF

WHEREFORE, Plaintiff Synopsys, Inc. prays that the Court enter judgment that:

(a) United States Patent No. 4,922,432 is invalid;

(b) United States Patent No. 4,922,432 is unenforceable;

(c) United States Patent No. 4,922,432 is not infringed by Synopsys and its customers;

(d) United States Patent No. 5,197,016 is invalid;

(e) United States Patent No. 5,197,016 is unenforceable;

(f) United States Patent No. 5,197,016 is not infringed by Synopsys and its customers;

(g) For a preliminary and permanent injunction enjoining and restraining Ricoh, and its respective officers, agents, servants, employees and attorneys, and all persons acting in concert with them, and each of them:

(i) From making any claims to any person or entity that any Synopsys logic synthesis product infringes either of the '432 or '016 patents;

(ii) From interfering with or threatening to interfere with the manufacture, use, or sale of any Synopsys logic synthesis product; and

(iii) From instituting or prosecuting any lawsuit or proceeding, placing in issue the right of Synopsys, its distributors, customers, predecessors, or assigns to make, use, or sell any logic synthesis product;

(h) Synopsys be awarded damages in accordance with its proof at trial;

(i) Synopsys be awarded its costs and reasonable attorneys' fees incurred herein; and

(j) Such other and further relief be granted as the Court deems just and proper.

JURY DEMAND

Synopsys demands a trial by jury on all issues so triable.

Dated: May 15, 2003

Respectfully submitted,

HOWREY SIMON ARNOLD & WHITE, LLP

By: _____

Teresa M. Corbin
Christopher Kelley
Erik K. Moller
Attorneys for Plaintiff
Synopsys, Inc.

EXHIBIT A

United States Patent [19]

Kobayashi et al.

[11] **Patent Number: 4,922,432**

[45] **Date of Patent: May 1, 1990**

[54] **KNOWLEDGE BASED METHOD AND APPARATUS FOR DESIGNING INTEGRATED CIRCUITS USING FUNCTIONAL SPECIFICATIONS**

[75] Inventors: **Hideaki Kobayashi**, Columbia, S.C.; **Masahiro Shindo**, Osaka, Japan

[73] Assignees: **International Chip Corporation**, Columbia, S.C.; **Ricoh Company, Ltd.**, Tokyo, Japan

[21] Appl. No.: 143,821

[22] Filed: Jan. 13, 1988

[51] Int. Cl.[5] G06F 15/60

[52] U.S. Cl. 364/490; 364/489; 364/488; 364/521

[58] Field of Search 364/488–491, 364/521, 300, 513

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,635,208	1/1987	Coleby et al.	364/491
4,638,442	1/1987	Bryant et al.	364/489
4,648,044	3/1987	Hardy et al.	364/513
4,651,284	3/1987	Watanabe et al.	364/491
4,656,603	4/1987	Dunn	364/488
4,658,370	4/1987	Erman et al.	364/513
4,675,829	6/1987	Clemenson	364/513
4,700,317	10/1987	Watanabe et al.	364/521
4,703,435	10/1987	Darringer et al.	364/488
4,803,636	2/1989	Nishiyama et al.	364/491

FOREIGN PATENT DOCUMENTS

1445914	8/1976	United Kingdom	364/490

OTHER PUBLICATIONS

"Verifying Compiled Silicon", by E. K. cheng, VLSI Design, Oct. 1984, pp. 1–4.

"CAD System for IC Design", by M. E. Daniel et al., IEEE Trans. on Computer-Aided Design of Integrated Circuits & Systems, vol. CAD–1, No. 1, Jan. 1982, pp. 2–12.

"An Overview of Logic Synthesis System", by L. Trevillyan, 24th ACM/IEEE Design Automation Conference, 1978, pp. 166–172.

"Methods Used in an Automatic Logic Design Generator", by T. D. Friedman et al., IEEE Trans. on Computers, vol. C–18, No. 7, Jul. 1969, pp. 593–613.

"Experiments in Logic Synthesis", by J. A. Darringer, IEEE ICCC, 1980.

"A Front End Graphic Interface to First Silicon Compiler", by J. H. Nash, EDA 84, Mar. 1984.

"quality of Designs from An Automatic Logic Generator", by T. D. Friedman et al., IEEE 7th DA Conference, 1970, pp. 71–89.

"A New Look at Logic Synthesis", by J. A. Darringer et al., IEEE 17th D. A. Conference 1980, pp. 543–548.

Trevillyan—Trickey, H., Flamel: *A High Level Hardward Compiler*, IEEE Transactions On Computer Aided Design, Mar. 1987, pp. 259–269.

Parker et al., The *CMU Design Automation System—An Example of Automated Data Path Design*, Proceedings Of The 16th Design Automation Conference, Las Vegas, Nev., 1979, pp. 73–80.

An Engineering Approach to Digital Design, William I. Fletcher, Prentice-Hall, Inc., pp. 491–505.

Primary Examiner—Felix D. Gruber

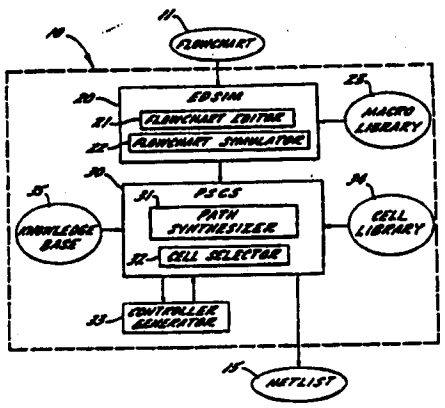
Assistant Examiner—V. N. Trans

Attorney, Agent, or Firm—Bell, Seltzer, Park & Gibson

[57] **ABSTRACT**

The present invention provides a computer-aided design system and method for designing an application specific integrated circuit which enables a user to define functional architecture independent specifications for the integrated circuit and which translates the functional architecture independent specifications into the detailed information needed for directly producing the integrated circuit. The functional architecture independent specifications of the desired integrated circuit can be defined at the functional architecture independent level in a flowchart format. From the flowchart, the system and method uses artificial intelligence and expert systems technology to generate a system controller, to select the necessary integrated circuit hardware cells needed to achieve the functional specifications, and to generate data and control paths for operation of the integrated circuit. This list of hardware cells and their interconnection requirements is set forth in a netlist. From the netlist it is possible using known manual techniques or existing VLSI CAD layout systems to generate the detailed chip level topological information (mask data) required to produce the particular application specific integrated circuit.

20 Claims, 12 Drawing Sheets



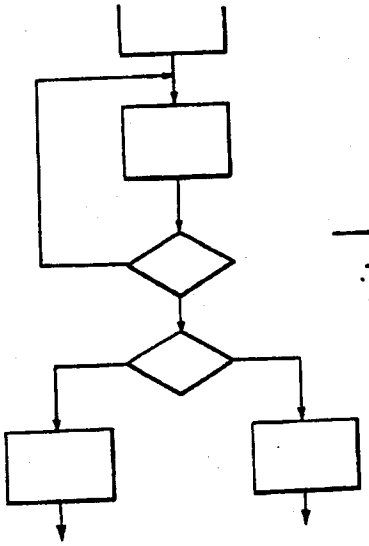


Fig. 1a.
FUNCTIONAL LEVEL

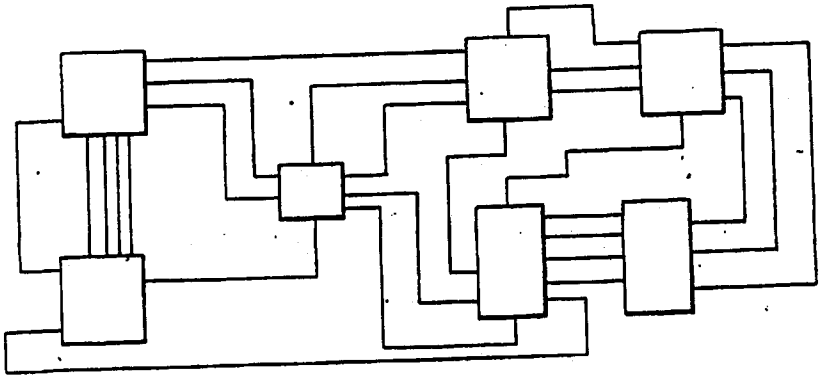


Fig. 1b.
STRUCTURAL LEVEL

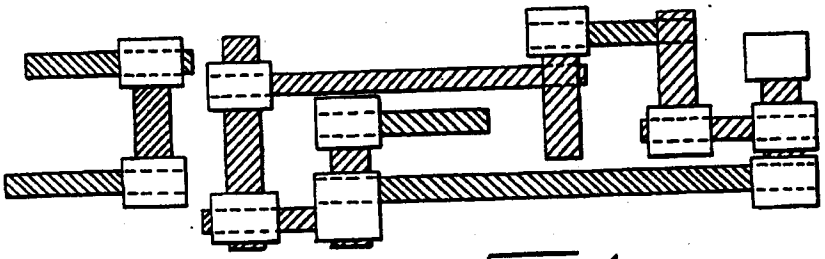


Fig. 1c.
PHYSICAL LAYOUT LEVEL

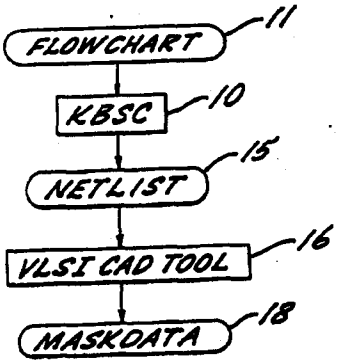

FLOWCHART
11
KBSC
10
NETLIST
15
VLSI CAD TOOL
16
MASKDATA
18

Fig. 2.

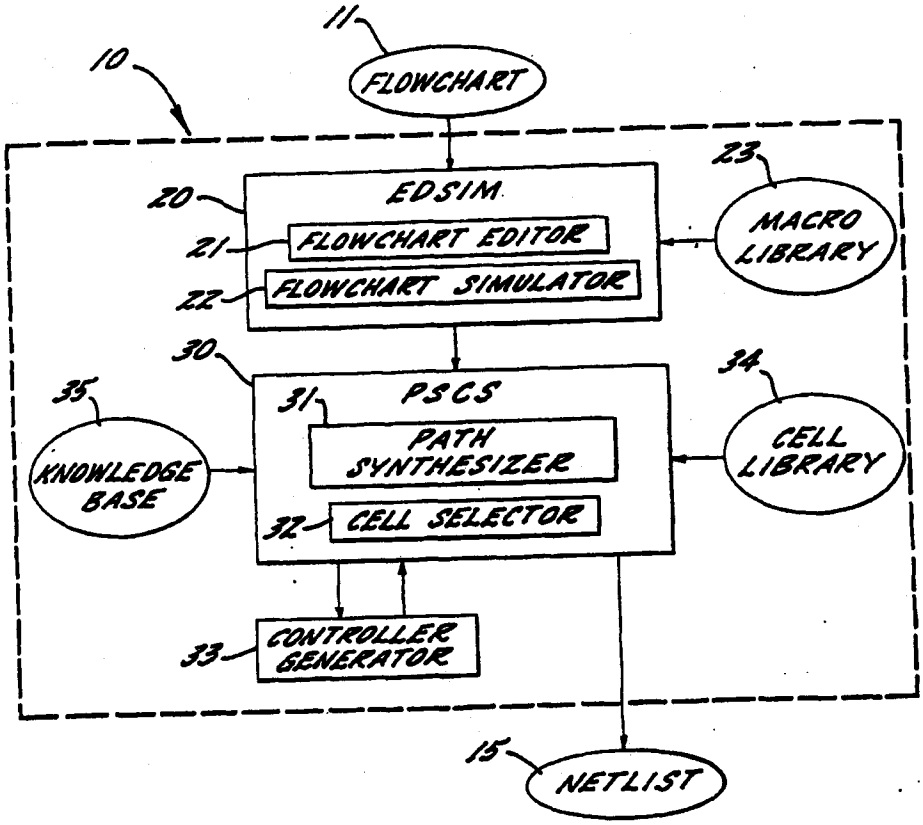

11
FLOWCHART
10
EDSIM
20
FLOWCHART EDITOR
21
FLOWCHART SIMULATOR
22
23
MACRO LIBRARY
30
PSCS
31
PATH SYNTHESIZER
32
CELL SELECTOR
33
CONTROLLER GENERATOR
35
KNOWLEDGE BASE
34
CELL LIBRARY
15
NETLIST

Fig. 3.

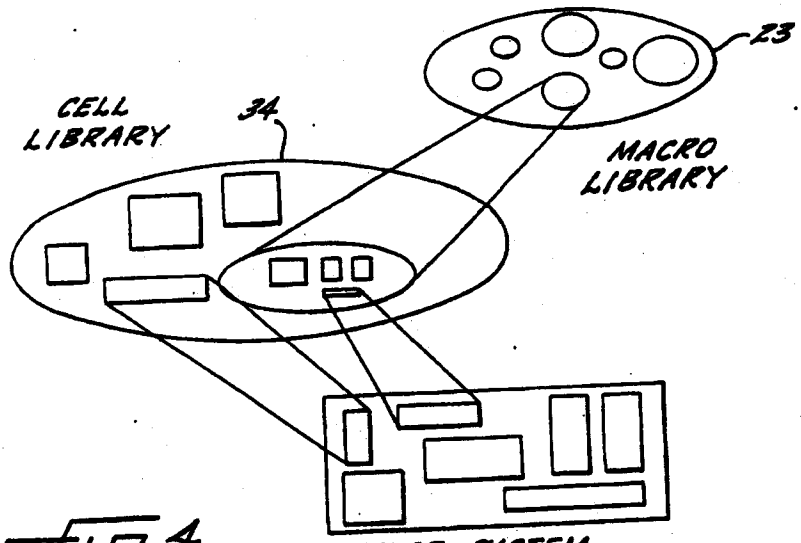

23
CELL LIBRARY
34
MACRO LIBRARY
VLSI SYSTEM

Fig. 4.

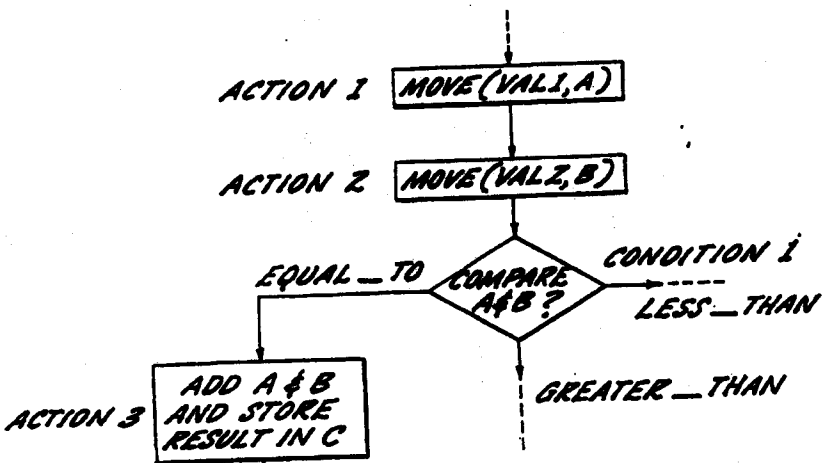

ACTION 1
MOVE(VAL1,A)
ACTION 2
MOVE(VAL2,B)
EQUAL_TO
COMPARE A&B?
CONDITION 1
LESS_THAN
GREATER_THAN
ACTION 3
ADD A&B AND STORE RESULT IN C

Fig. 5.

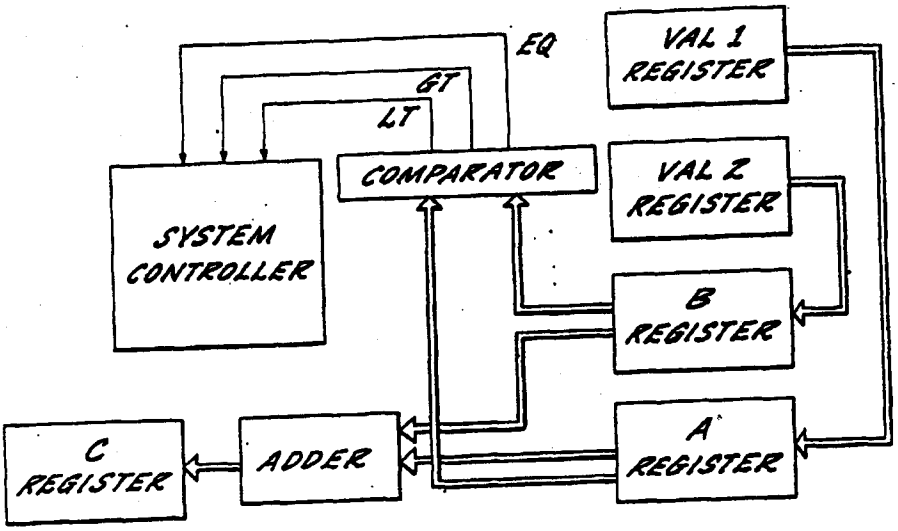

EQ
GT
LT
VAL 1 REGISTER
COMPARATOR
VAL 2 REGISTER
SYSTEM CONTROLLER
B REGISTER
A REGISTER
C REGISTER
ADDER

FIG. 6.

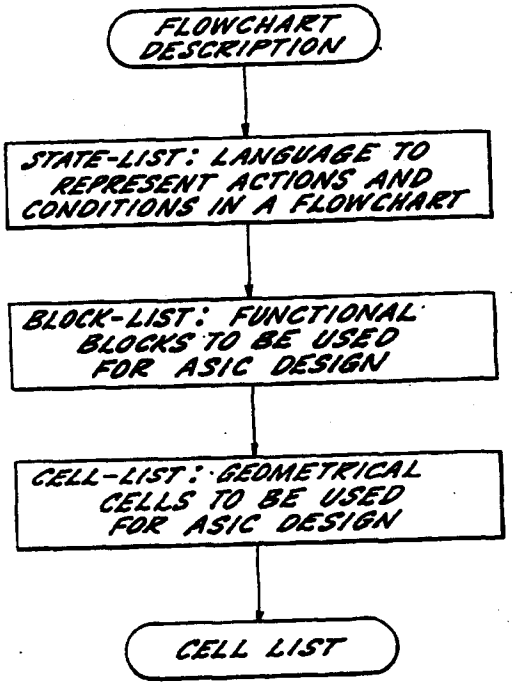

FLOWCHART DESCRIPTION
STATE-LIST: LANGUAGE TO REPRESENT ACTIONS AND CONDITIONS IN A FLOWCHART
BLOCK-LIST: FUNCTIONAL BLOCKS TO BE USED FOR ASIC DESIGN
CELL-LIST: GEOMETRICAL CELLS TO BE USED FOR ASIC DESIGN
CELL LIST

FIG. 9.

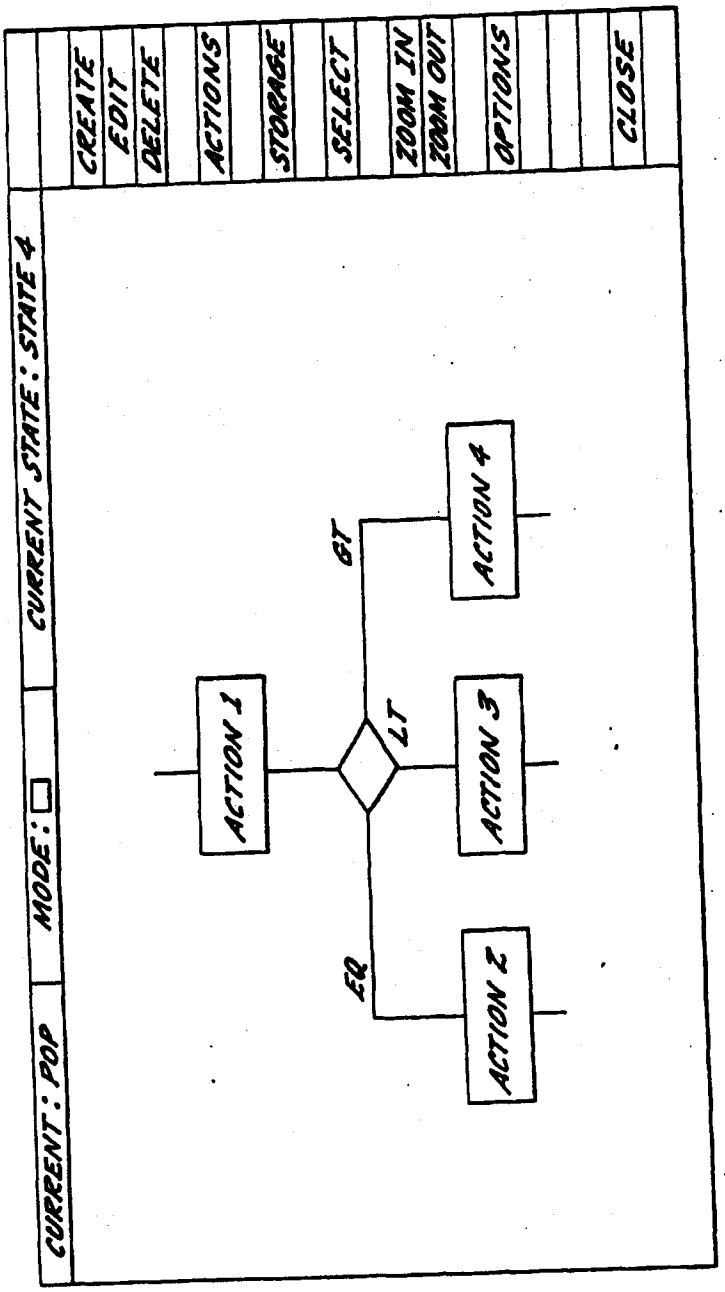

CREATE
EDIT
DELETE
ACTIONS
STORAGE
SELECT
ZOOM IN
ZOOM OUT
OPTIONS
CLOSE
CURRENT STATE: STATE 4
MODE: □
CURRENT: POP
ACTION 1
GT
LT
EQ
ACTION 4
ACTION 3
ACTION 2

FIG. 7.

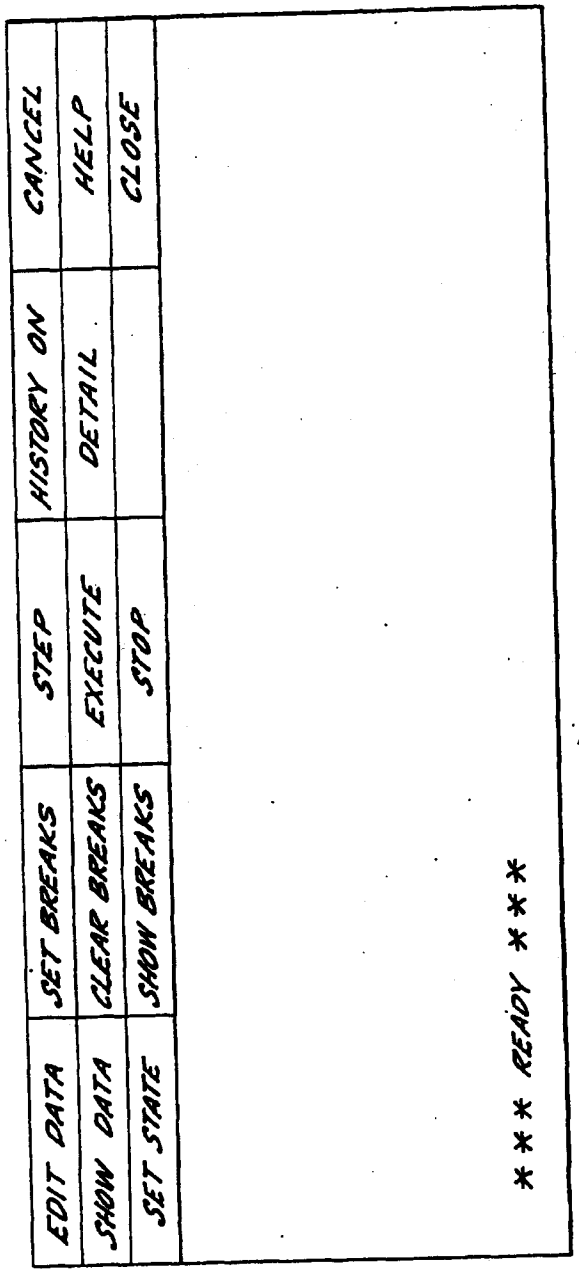

EDIT DATA
SHOW DATA
SET STATE
SET BREAKS
CLEAR BREAKS
SHOW BREAKS
STEP
EXECUTE
STOP
HISTORY ON
DETAIL
CANCEL
HELP
CLOSE
*** READY ***

FIG. 8.

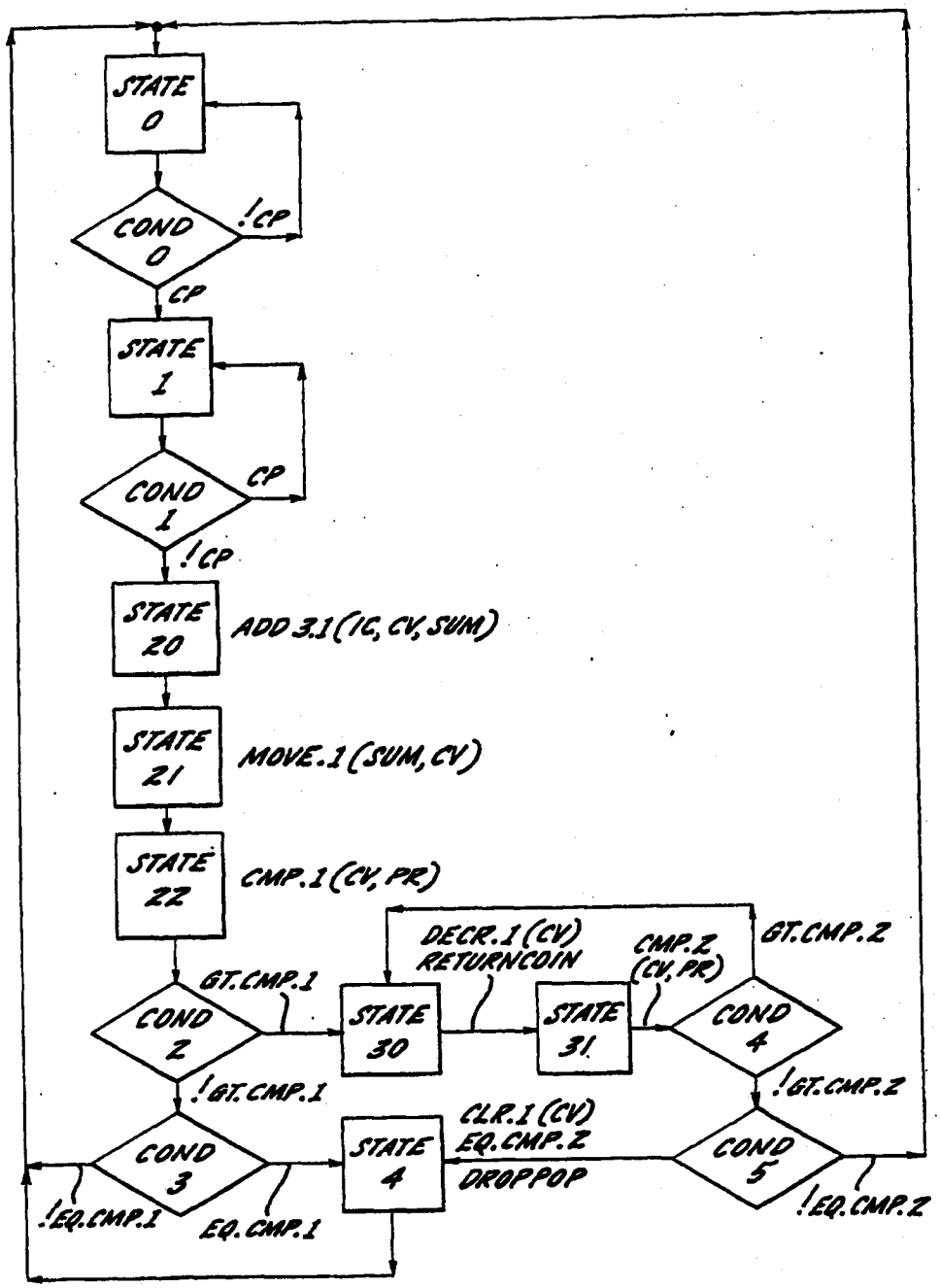

STATE 0
COND 0
!CP
CP
STATE 1
COND 1
CP
!CP
STATE 20
ADD 3.1 (IC, CV, SUM)
STATE 21
MOVE.1 (SUM, CV)
STATE 22
CMP.1 (CV, PR)
GT.CMP.1
COND 2
!GT.CMP.1
STATE 30
DECR.1 (CV)
RETURNCOIN
STATE 31
CMP.2 (CV, PR)
GT.CMP.2
COND 4
!GT.CMP.2
COND 3
!EQ.CMP.1
EQ.CMP.1
STATE 4
CLR.1 (CV)
EQ.CMP.2
DROPPOP
COND 5
!EQ.CMP.2

FIG. 10.

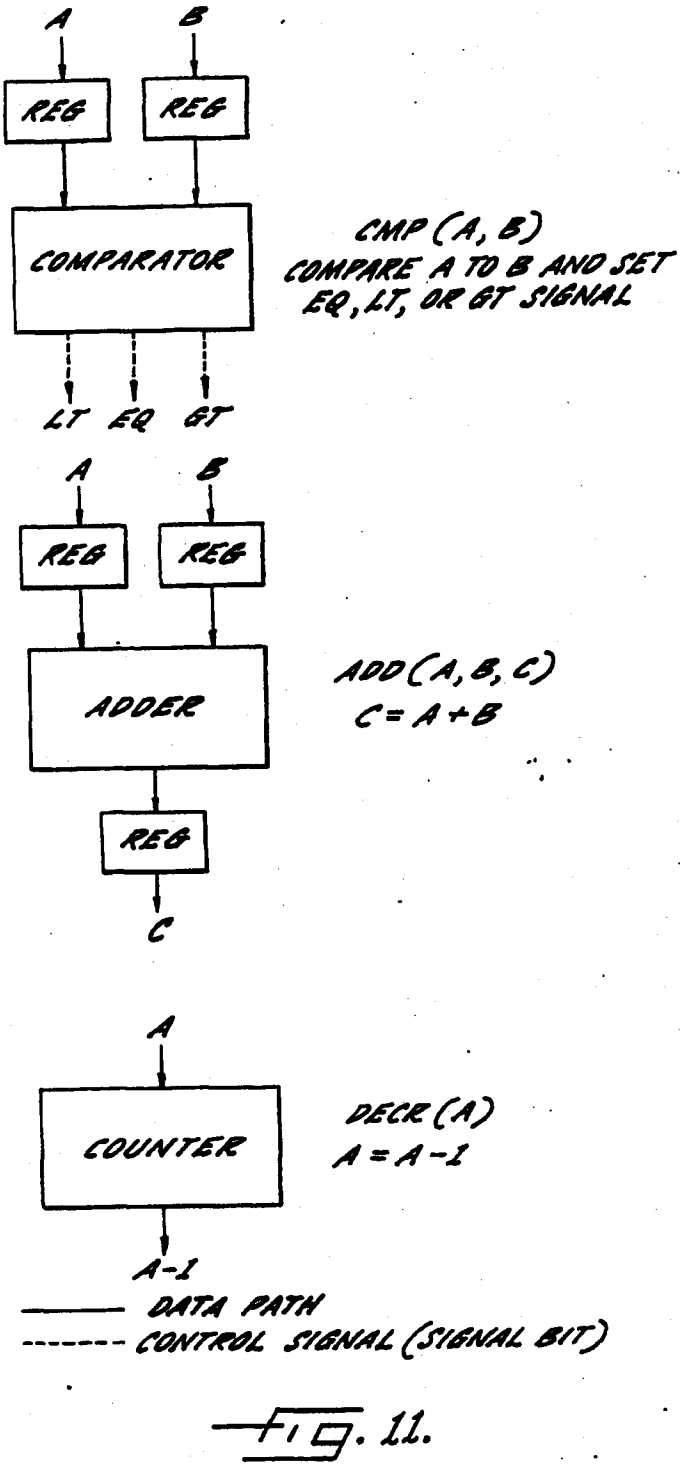

A
B
REG
REG
COMPARATOR
CMP (A, B)
COMPARE A TO B AND SET
EQ, LT, OR GT SIGNAL
LT
EQ
GT
A
B
REG
REG
ADDER
ADD (A, B, C)
C = A + B
REG
C
A
COUNTER
DECR (A)
A = A - 1
A-1
DATA PATH
CONTROL SIGNAL (SIGNAL BIT)

Fig. 11.

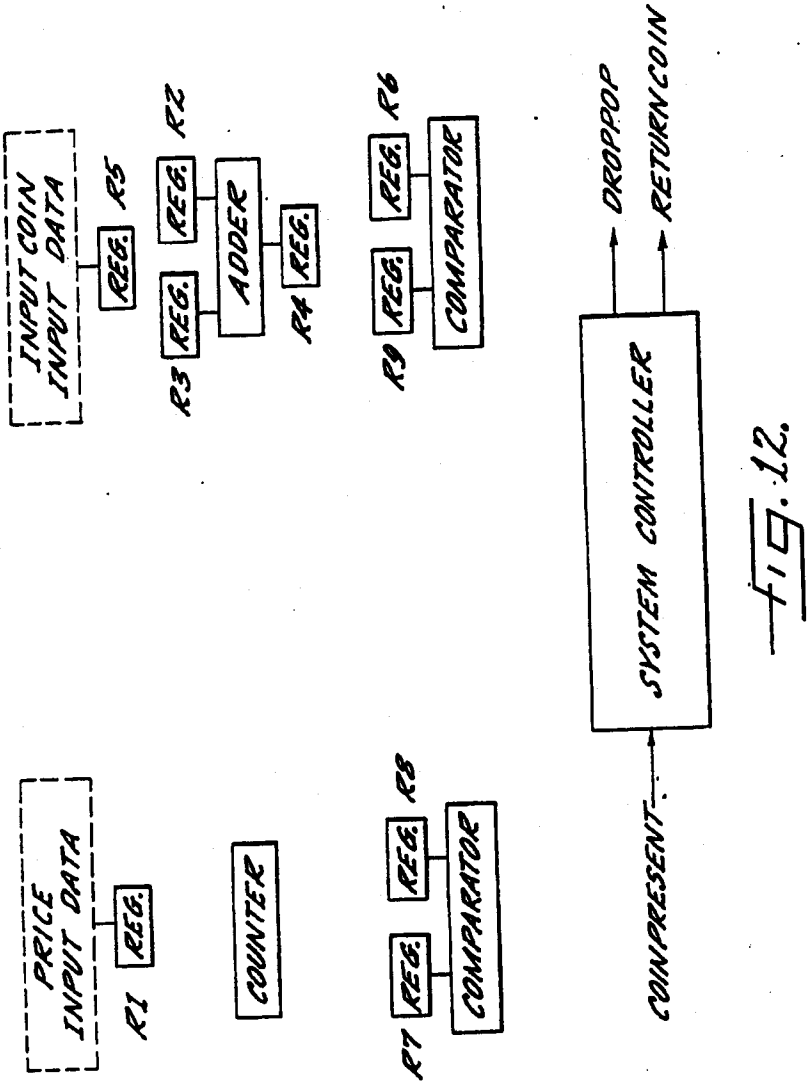

INPUT COIN
INPUT DATA
REG.
R5
REG.
R2
REG.
R3
ADDER
R4
REG.
REG.
R6
REG.
R9
COMPARATOR
DROPPOP
RETURN COIN
SYSTEM CONTROLLER
COINPRESENT
PRICE
INPUT DATA
REG.
R1
COUNTER
REG.
R8
REG.
R7
COMPARATOR

FIG. 12.

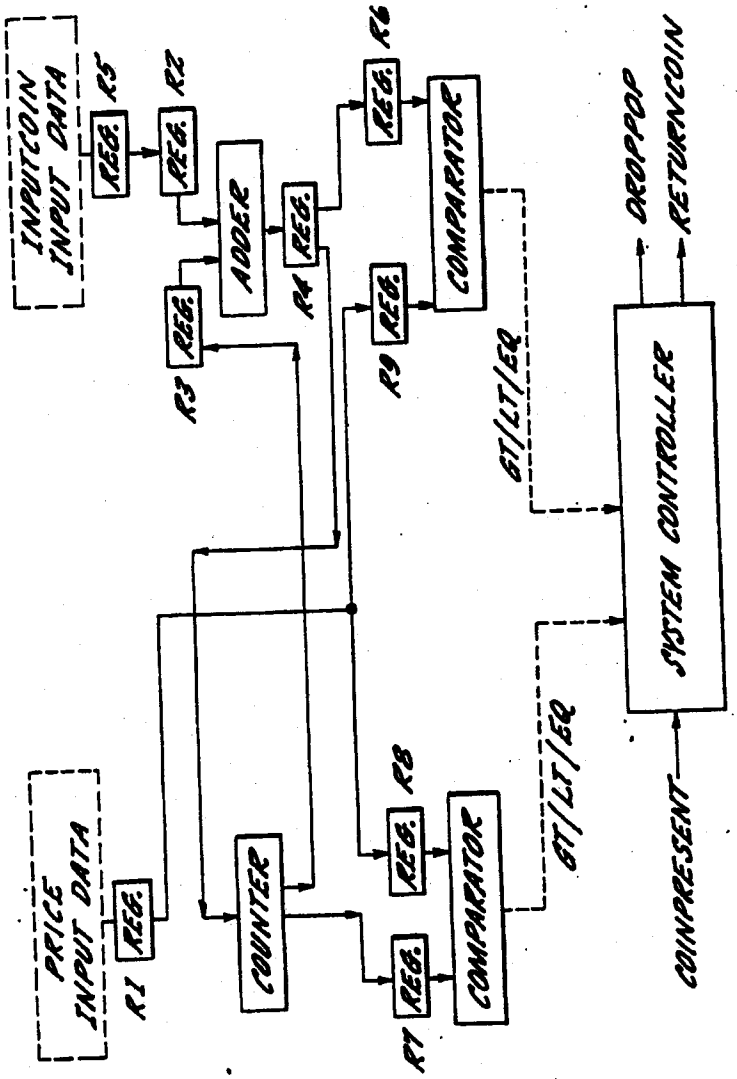

INPUT COIN
INPUT DATA
REG.
R5
R2
REG.
ADDER
R6
REG.
COMPARATOR
R4
REG.
R3
REG.
R9
REG.
GT/LT/EQ
DROPPOP
RETURNCOIN
SYSTEM CONTROLLER
PRICE
INPUT DATA
R1
REG.
COUNTER
R8
REG.
COMPARATOR
GT/LT/EQ
R7
REG.
COINPRESENT

FIG. 13.

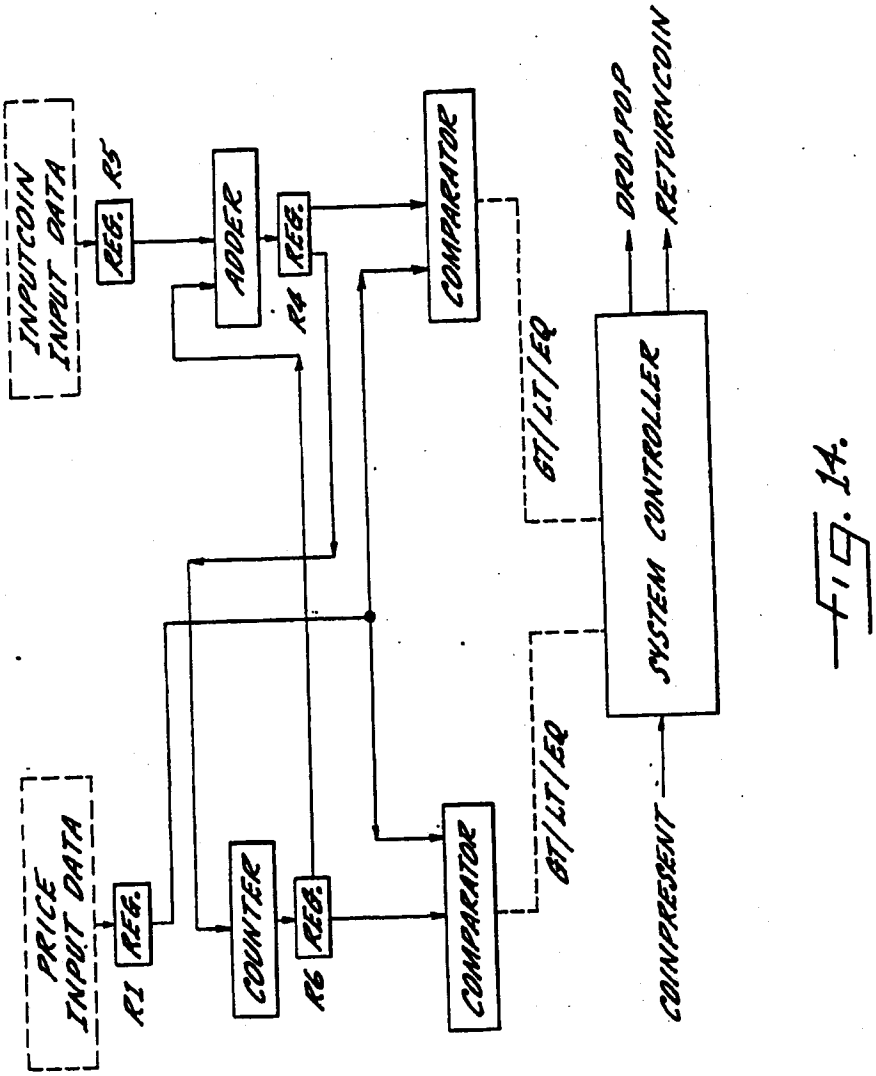

INPUTCOIN INPUT DATA
REG.
R5
ADDER
REG.
R4
COMPARATOR
DROPPOP
RETURNCOIN
GT/LT/EQ
SYSTEM CONTROLLER
PRICE INPUT DATA
REG.
R1
COUNTER
REG.
R6
COMPARATOR
GT/LT/EQ
COINPRESENT

FIG. 14.

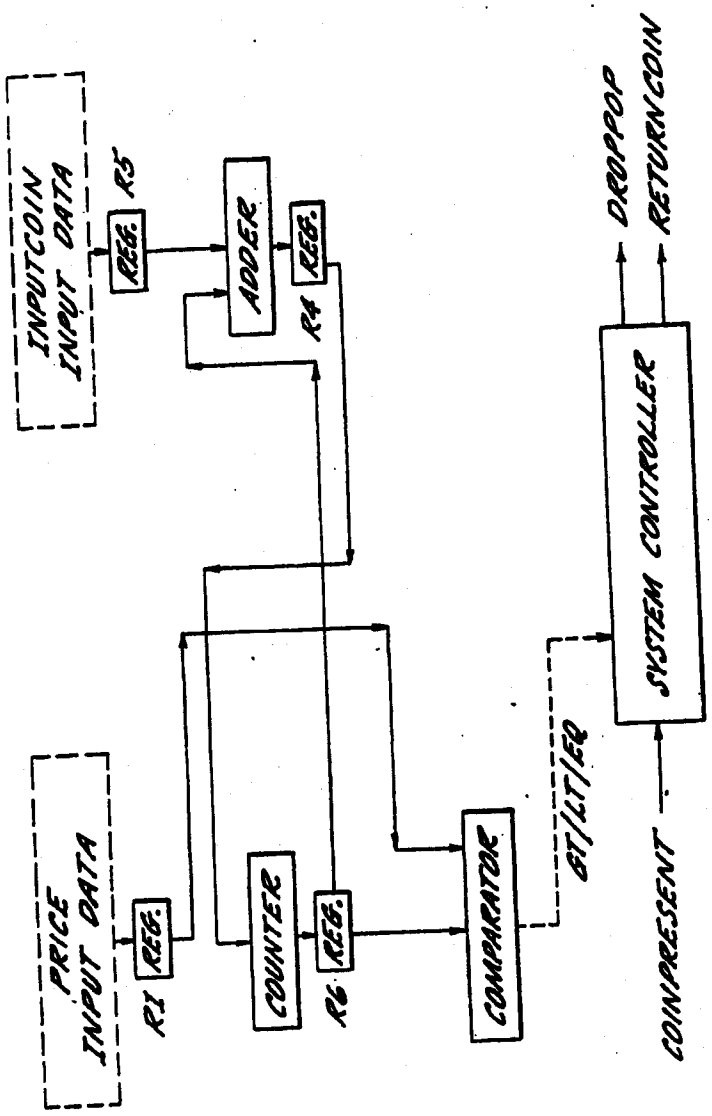

INPUTCOIN INPUT DATA
REG.
R5
ADDER
REG.
R4
DROPPOP
RETURNCOIN
SYSTEM CONTROLLER
PRICE INPUT DATA
REG.
R1
COUNTER
REG.
R6
COMPARATOR
GT/LT/EQ
COINPRESSENT

FIG. 15.

KNOWLEDGE BASED METHOD AND APPARATUS FOR DESIGNING INTEGRATED CIRCUITS USING FUNCTIONAL SPECIFICATIONS

FIELD AND BACKGROUND OF THE INVENTION

This invention relates to the design of integrated circuits, and more particularly relates to a computer-aided method and apparatus for designing integrated circuits.

An application specific integrated circuit (ASIC) is an integrated circuit chip designed to perform a specific function, as distinguished from standard, general purpose integrated circuit chips, such as microprocessors, memory chips, etc. A highly skilled design engineer having specialized knowledge in VLSI circuit design is ordinarily required to design a ASIC. In the design process, the VLSI design engineer will consider the particular objectives to be accomplished and tasks to be performed by the integrated circuit and will create structural level design specifications which define the various hardware components required to perform the desired function, as well as the interconnection requirements between these components. A system controller must also be designed for synchronizing the operations of these components. This requires an extensive and all encompassing knowledge of the various hardware components required to achieve the desired objectives, as well as their interconnection requirements, signal level compatibility, timing compatibility, physical layout, etc. At each design step, the designer must do tedious analysis. The design specifications created by the VLSI design engineer may, for example, be in the form of circuit schematics, parameters or specialized hardware description languages (HDLs).

From the structural level design specifications, the description of the hardware components and interconnections is converted to a physical chip layout level description which describes the actual topological characteristics of the integrated circuit chip. This physical chip layout level description provides the mask data needed for fabricating the chip.

Due to the tremendous advances in very large scale integration (VLSI) technology, highly complex circuit systems are being built on a single chip. With their complexity and the demand to design custom chips at a faster rate, in large quantities, and for an ever increasing number of specific applications, computer-aided design (CAD) techniques need to be used. CAD techniques have been used with success in design and verification of integrated circuits, at both the structural level and at the physical layout level. For example, CAD systems have been developed for assisting in converting VLSI structural level descriptions of integrated circuits into the physical layout level topological mask data required for actually producing the chip. Although the presently available computer-aided design systems greatly facilitate the design process, the current practice still requires highly skilled VLSI design engineers to create the necessary structural level hardware descriptions.

There is only a small number of VLSI designers who possess the highly specialized skills needed to create structural level integrated circuit hardware descriptions. Even with the assistance of available VLSI CAD tools, the design process is time consuming and the probability of error is also high because of human involvements. There is a very significant need for a better and more cost effective way to design custom integrated circuits.

SUMMARY OF THE INVENTION

In accordance with the present invention a CAD (computer-aided design) system and method is provided which enables a user to define the functional requirements for a desired target integrated circuit, using an easily understood functional architecture independent level representation, and which generates therefrom the detailed information needed for directly producing an application specific integrated circuit (ASIC) to carry out those specific functions. Thus, the present invention, for the first time, opens the possibility for the design and production of ASICs by designers, engineers and technicians who may not possess the specialized expert knowledge of a highly skilled VLSI design engineer.

The functional architecture independent specifications of the desired ASIC can be defined in a suitable manner, such as in list form or preferably in a flowchart format. The flowchart is a highly effective means of describing a sequence of logical operations, and is well understood by software and hardware designers of varying levels of expertise and training. From the flowchart (or other functional specifications), the system and method of the present invention translates the functional architecture independent specifications into structural an architecture specific level definition of an integrated circuit, which can be used directly to produce the ASIC. The structural level definition includes a list of the integrated circuit hardware cells needed to achieve the functional specifications. These cells are selected from a cell library of previously designed hardware cells of various functions and technical specifications. The system also generates data paths among the selected hardware cells. In addition, the present invention generates a system controller and control paths for the selected integrated circuit hardware cells. The list of hardware cells and their interconnection requirements may be represented in the form of a netlist. From the netlist it is possible using either known manual techniques or existing VLSI CAD layout systems to generate the detailed chip level geometrical information (e.g. mask data) required to produce the particular application specific integrated circuit in chip form.

The preferred embodiment of the system and method of the present invention which is described more fully hereinafter is referred to as a Knowledge Based Silicon Compiler (KBSC). The KBSC is an ASIC design methodology based upon artificial intelligence and expert systems technology. The user interface of KBSC is a flowchart editor which allows the designer to represent VLSI systems in the form of a flowchart. The KBSC utilizes a knowledge based expert system, with a knowledge base extracted from expert ASIC designers with a high level of expertise in VLSI design to generate from the flowchart a netlist which describes the selected hardware cells and their interconnection requirements.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be better understood by reference to the detailed description which follows, taken in connection with the accompanying drawings, in which

FIG. 1*a* illustrates a functional level design representation of a portion of a desired target circuit, shown in the form of a flowchart;

FIG. 1*b* illustrates a structural level design representation of an integrated circuit;

FIG. 1*c* illustrates a design representation of a circuit at a physical layout level, such as would be utilized in the fabrication of an integrated circuit chip;

FIG. 2 is a block schematic diagram showing how integrated circuit mask data is created from flowchart descriptions by the KBSC system of the present invention;

FIG. 3 is a somewhat more detailed schematic illustration showing the primary components of the KBSC system;

FIG. 4 is a schematic illustration showing how the ASIC design system of the present invention draws upon selected predefined integrated circuit hardware cells from a cell library;

FIG. 5 is an example flowchart defining a sequence of functional operations to be performed by an integrated circuit;

FIG. 6 is a structural representation showing the hardware blocks and interconnection requirements for the integrated circuit defined in FIG. 5;

FIG. 7 is an illustration of the flowchart editor window;

FIG. 8 is an illustration of the flowchart simulator window;

FIG. 9 is an illustration of the steps involved in cell list generation;

FIG. 10 is an example flowchart for a vending machine system;

FIG. 11 illustrates the hardware components which correspond to each of the three macros used in the flowchart of FIG. 10;

FIG. 12 is an initial block diagram showing the hardware components for an integrated circuit as defined in the flowchart of FIG. 10;

FIG. 13 is a block diagram corresponding to FIG. 12 showing the interconnections between blocks;

FIG. 14 is a block diagram corresponding to FIG. 13 after register optimization; and

FIG. 15 is a block diagram corresponding to FIG. 14 after further optimization.

DETAILED DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

FIGS. 1*a*, 1*b* and 1*c* illustrate three different levels of representing the design of an integrated circuit. FIG. 1*a* shows a functional (or behavioral) representation architecture independent in the form of a flowchart. A flowchart is a graphic representation of an algorithm and consists of two kinds of blocks or states, namely actions and conditions (decisions). Actions are conventionally represented in the flowchart by a rectangle or box, and conditions are represented by a diamond. Transitions between actions and conditions are represented by lines with arrows. FIG. 1*b* illustrates a structural (or logic) level representation of an integrated circuit. In this representation, blocks are used to represent integrated architecture specific circuit hardware components for performing various functions, and the lines interconnecting the blocks represent paths for the flow of data or control signals between the blocks. The blocks may, for example, represent hardware components such as adders, comparators, registers, system controllers, etc. FIG. 1*c* illustrates a physical layout level representation of an integrated circuit design, which provides the detailed mask data necessary to actually manufacture the devices and conductors which together comprise integrated circuit.

As noted earlier, the design of an integrated circuit at the structural level requires a design engineer with highly specialized skills and expertise in VLSI design. In the KBSC system of the present invention, however, integrated circuits can be designed at a functional level because the expertise in VLSI design is provided and applied by the invention. Allowing the designer to work with flowcharts instead of logic circuit schematics simplifies the task of designing custom integrated circuits, making it quicker, less expensive and more reliable. The designer deals with an algorithm using simple flowcharts at an architecture independent functional (behavioral) level, and needs to know only the necessary logical steps to complete a task, rather than the specific means for accomplishing the task. Designing with flowcharts requires less work in testing because flowcharts allow the designer to work much closer to the algorithm. On the other hand, previously existing VLSI design tools require the designer to represent an algorithm with complex circuit schematics at a structural level, therefore requiring more work in testing. Circuit schematics make it harder for the designer to cope with the algorithm function which needs to be incorporated into the target design because they intermix the hardware and functional considerations. Using flowcharts to design custom integrated circuits will allow a large number of system designers to access VLSI technology, where previously only a small number of designers had the knowledge and skills to create the necessary structural level hardware descriptions.

The overall system flow is illustrated in FIG. 2. The user enters the functional specifications of the circuit into the knowledge based silicon compiler (KBSC) 10 in the form of a flowchart 11. The KBSC 10 then generates a netlist 15 from the flowchart. The netlist 15 includes a custom generated system controller, all other hardware cells required to implement the necessary operations, and interconnection information for connecting the hardware cells and the system controller. The netlist can be used as input to any existing VLSI layout and routing tool 16 to create mask data 18 for geometrical layout.

System Overview

The primary elements or modules which comprise the KBSC system are shown in FIG. 3. In the embodiment illustrated and described herein, these elements or modules are in the form of software programs, although persons skilled in the appropriate art will recognize that these elements can easily be embodied in other forms, such as in hardware.

Referring more particularly to FIG. 3, it will be seen that the KBSC system 10 includes a program 20 called EDSIM, which comprises a flowchart editor 21 for creating and editing flowcharts and a flowchart simulator 22 for simulation and verification of flowcharts. Actions to be performed by each of the rectangles represented in the flowchart are selected from a macro library 23. A program 30 called PSCS (path synthesizer and cell selector) includes a data and control path synthesizer module 31, which is a knowledge based system for data and control path synthesis. PSCS also includes a cell selector 32 for selecting the cells required for system design. The cell selector 32 selects from a cell

library 34 of previously designed hardware cells the appropriate cell or cells required to perform each action and condition represented in the flowchart. A controller generator 33 generates a custom designed system controller for controlling the operations of the other hardware cells. The knowledge base 35 contains ASIC design expert knowledge required for data path synthesis and cell selection. Thus, with a functional flowchart input, PSCS generates a system controller, selects all other hardware cells, generates data and control paths, and generates a netlist describing all of this design information.

The KBSC system employs a hierarchal cell selection ASIC design approach, as is illustrated in FIG. 4. Rather than generating every required hardware cell from scratch, the system draws upon a cell library 34 of previously designed, tested and proven hardware cells of various types and of various functional capabilities with a given type. The macro library 23 contains a set of macros defining various actions which can be specified in the flowchart. For each macro function in the macro library 23 there may be several hardware cells in the cell library 34 of differing geometry and characteristics capable of performing the specified function. Using a rule based expert system with a knowledge base 35 extracted from expert ASIC designers, the KBSC system selects from the cell library 34 the optimum cell for carrying out the desired function.

Referring again to FIG. 3, the cells selected by the cell selector 32, the controller information generated by the controller generator 33 and the data and control paths generated by the data/control path synthesizer 31 are all utilized by the PSCS program 30 to generate the netlist 15. The netlist is a list which identifies each block in the circuit and the interconnections between the respective inputs and outputs of each block. The netlist provides all the necessary information required to produce the integrated circuit. Computer-aided design systems for cell placement and routing are commercially available which will receive netlist data as input and will lay out the respective cells in the chip, generate the necessary routing, and produce mask data which can be directly used by a chip foundry in the fabrication of integrated circuits.

System Requirements

The KBSC system can be operated on a suitable programed general purpose digital computer. By way of example, one embodiment of the system is operated in a work station environment such as Sun3 and VAXStation-II/GPX Running UNIX Operating System and X Window Manager. The work station requires a minimum of 8 megabytes of main storage and 20 megabytes of hard disk space. The monitor used is a color screen with 8-bit planes. The software uses C programming language and INGRES relational data base.

The human interface is mainly done by the use of pop up menus, buttons, and a special purpose command language. The permanent data of the integrated circuit design are stored in the data base for easy retrieval and update. Main memory stores the next data temporarily, executable code, design data (flowchart, logic, etc.), data base (cell library), and knowledge base. The CPU performs the main tasks of creating and simulating flowcharts and the automatic synthesis of the design.

Flowchart Example

To describe the mapping of a flowchart to a netlist, consider an example flowchart shown in FIG. 5, which is of part of a larger overall system. In this illustrative flowchart, two variables, VAL1 and VAL2 are compared and if they are equal, they are added together. In this instance, the first action (Action 1) involves moving the value of variable VALI to register A. The second action comprises moving the value of variable VAL2 to register B. Condition 1 comprises comparing the values in registers A and B. Action 3 comprises adding the values of registers A and B and storing the result in register C.

In producing an integrated circuit to carry out the function defined in FIG. 5, the KBSC maps the flowchart description of the behavior of the system to interconnection requirements between hardware cells. The hardware cells are controlled by a system controller which generates all control signals. There are two types of variables involved in a system controller:

(1) Input variables: These are generated by hardware cells, and/or are external input to the controller. These correspond to conditions in the flowchart.

(2) Output variables: These are generated by the system controller and correspond to actions in the flowchart.

FIG. 6 illustrates the results of mapping the flowchart of FIG. 5 onto hardware cells. The actions and the conditions in the flowchart are used for cell selection and data and control path synthesis. The VALI register and VAL2 register and the data paths leading therefrom have already been allocated in actions occurring before Action 1 in our example. Action 1 causes generation of the data register A. Similarly, Action 2 causes the allocation of data register B. The comparator is allocated as a result of the comparison operation in Condition 1. The comparison operation is accomplished by (1) selecting a comparator cell, (2) mapping the inputs of the comparator cell to registers A and B, (3) generating data paths to connect the comparator with the registers A and B and (4) generating input variables corresponding to equal to, greater than, and less than for the system controller. Similarly the add operation in Action 3 causes selection of the adder cell, mapping of the adder parameters to the registers and creating the data paths.

Following this methodology, a block list can be generated for a given flowchart. This block list consists of a system controller and as many other blocks as may be required for performing the necessary operations. The blocks are connected with data paths, and the blocks are controlled by the system controller through control paths. These blocks can be mapped to the cells selected from a cell library to produce a cell list.

Interactive Flowchart Editor and Simulator

The creation and verification of the flowchart is the first step in the VLSI design methodology. The translation from an algorithm to an equivalent flowchart is performed with the Flowchart Editor 21 (FIG. 3). The verification of the edited flowchart is performed by the Flowchart Simulator 22 The Flowchart Editor and Simulator are integrated into one working environment for interactive flowchart editing, with a designer friendly interface.

EDSIM is the program which contains the Flowchart Editor 21 and the Flowchart Simulator 22. It also provides functions such as loading and saving flow-

charts. EDSIM will generate an intermediate file, called a statelist, for each flowchart. This file is then used by the PSCS program 30 to generate a netlist.

Flowchart Editor

The Flowchart Editor 21 is a software module used for displaying, creating, and editing the flowchart. This module is controlled through the flowchart editing window illustrated in FIG. 7. Along with editing functions the Flowchart Editor also provides checking of design errors.

The following is a description of the operations of the Flowchart Editor. The main editing functions include, create, edit, and delete states, conditions, and transitions. The create operation allows the designer to add a new state, condition, or transitions to a flowchart. Edit allows the designer to change the position of a state, condition or transition, and delete allows the designer to remove a state, condition or transition from the current flowchart. States which contain actions are represented by boxes, conditions are represented by diamonds, and transitions are represented by lines with arrows showing the direction of the transition.

Edit actions allows the designer to assign actions to each box. These actions are made up of macro names and arguments. An example of arguments is the setting and clearing of external signals. A list of basic macros available in the macro library 23 is shown in Table 1.

TABLE 1

Macro	Description
ADD (A,B,C)	C = A + B
SUB (A,B,C)	C = A − B
MULT (A,B,C)	C = A * B
DIV (A,B,C)	C = A div B
DECR (A)	A = A − 1
INCR (A)	A = A + 1
CLR (A)	A = 0
REG (A,B)	B = A
CMP (A,B)	Compare A to B and set EQ,LT,GT signals
CMP0 (A)	Compare A to 0 and set EQ,LT,GT signals
NEGATE (A)	A = NOT (A)
MOD (A,B,C)	C = A Modulus B
POW (A,B,C)	C = A B
DC2 (A,S1,S2,S3,S4)	Decode A into S1,S2,S3,S4
EC2 (S1,S2,S3,S4,A)	Encode S1,S2,S3,S4 into A
MOVE (A,B)	B = A
CALL sub-flowchart (A,B, . . .)	Call a sub-flowchart. Pass A,B . . .
START (A,B, . . .)	Beginning state of a sub-flowchart
STOP (A,B . . .)	Ending state of a sub-flowchart

The Flowchart Editor also provides a graphical display of the flowchart as the Flowchart Simulator simulates the flowchart. This graphical display consists of boxes, diamonds, and lines as shown in FIG. 7. All are drawn on the screen and look like a traditional flowchart. By displaying the flowchart on the screen during simulation it allows the designer to design and verify the flowchart at the same time.

Flowchart Simulator

The Flowchart Simulator 22 is a software module used for simulating flowcharts. This module is controlled through the simulator window illustrated in FIG. 8. The Flowchart Simulator simulates the transitions between states and conditions in a flowchart. The following is a list of the operations of the Flowchart Simulator:

edit data—Change the value of a register or memory.
set state—Set the next state to be simulated.
set detail or summary display—Display summary or detail information during simulation.
set breaks—Set a breakpoint.
clear breaks—Clear all breakpoints.
show breaks—Display current breakpoints.
step—Step through one transition.
execute—Execute the flowchart.
stop—Stop executing of the flowchart. history ON or history OFF—Set history recording on or off.
cancel—Cancel current operation.
help—Display help screen.
close—Close the simulator window.

The results of the simulation are displayed within the simulator window. Also the editor window will track the flowchart as it is being simulated. This tracking of the flowchart makes it easy to edit the flowchart when an error is found.

Cell Selection

The Cell Selector 32 is a knowledge based system for selecting a set of optimum cells from the cell library 34 to implement a VLSI system. The selection is based on functional descriptions in the flowchart, as specified by the macros assigned to each action represented in the flowchart. The cells selected for implementing a VLSI system depend on factors such as cell function, fabrication technology used, power limitations, time delays etc. The cell selector uses a knowledge base extracted from VLSI design experts to make the cell selection.

To design a VLSI system from a flowchart description of a user application, it is necessary to match the functions in a flowchart with cells from a cell library. This mapping needs the use of artificial intelligence techniques because the cell selection process is complicated and is done on the basis of a number of design parameters and constraints. The concept used for cell selection is analogous to that used in software compilation. In software compilation a number of subroutines are linked from libraries. In the design of VLSI systems, a functional macro can be mapped to library cell.

FIG. 4 illustrates the concept of hierarchical cell selection. The cell selection process is performed in two steps:

(1) selection of functional macros
(2) selection of geometrical cells

A set of basic macros is shown in Table 1. A macro corresponds to an action in the flowchart. As an example, consider the operation of adding A and B and storing the result in C. This function is mapped to the addition macro ADD(X, Y, Z). The flowchart editor and flowchart simulator are used to draw the rectangles, diamonds and lines of the flowchart, to assign a macro selected from the macro library 23 to each action represented in the flowchart, and to verify the functions in flowcharts. The flowchart is converted into an intermediate form (statelist) and input to the Cell Selector.

The Cell Selector uses a rule based expert system to select the appropriate cell or cells to perform each action. If the cell library has a number of cells with different geometries for performing the operation specified by the macro, then an appropriate cell can be selected on the basis of factors such as cell function, process technology used, time delay, power consumption, etc.

The knowledge base of Cell Selector 32 contains information (rules) relating to:

(1) selection of macros
(2) merging two macros

(3) mapping of macros to cells
(4) merging two cells
(5) error diagnostics

The above information is stored in the knowledge base 35 as rules.

Cell List Generation

FIG. 9 shows the cell list generation steps. The first step of cell list generation is the transformation of the flowchart description into a structure that can be used by the Cell Selector. This structure is called the statelist. The blocklist is generated from the statelist by the inference engine. The blocklist contains a list of the functional blocks to be used in the integrated circuit. Rules of the following type are applied during this stage.

map arguments to data paths
map actions to macros
connect these blocks

Rules also provide for optimization and error diagnostics at this level.

The cell selector maps the blocks to cells selected from the cell library 34. It selects an optimum cell for a block. This involves the formulation of rules for selecting appropriate cells from the cell library. Four types of information are stored for each cell. These are:

(1) functional level information: description of the cell at the register transfer level.

(2) logic level information: description in terms of flip-flops and gates.

(3) circuit level information: description at the transistor level.

(4) Layout level information: geometrical mask level specification.

The attributes of a cell are:

cell name
description
function
width
height
status
technology
minimum delay
typical delay
maximum delay
power
file
designer
date
comment
inspector

In the cell selection process, the above information can be used. Some parameters that can be used to map macros to cells are:

(1) name of macro
(2) function to be performed
(3) complexity of the chip
(4) fabrication technology
(5) delay time allowed
(6) power consumption
(7) bit size of macro data paths

Netlist Generation

The netlist is generated after the cells have been selected by PSCS. PSCS also uses the macro definitions for connecting the cell terminals to other cells. PSCS uses the state-to-state transition information from an intermediate form representation of a flowchart (i.e. the statelist) to generate a netlist. PSCS contains the following knowledge for netlist generation:

(1) Data path synthesis
(2) Data path optimization
(3) Macro definitions
(4) Cell library
(5) Error detection and correction

The above information is stored in the knowledge base 35 as rules. Knowledge engineers help in the formulation of these rules from ASIC design experts. The macro library 23 and the cell library 34 are stored in a database of KBSC.

A number of operations are performed by PSCS. The following is a top level description of PSCS operations:

(1) Read the flowchart intermediate file and build a statelist.
(2) current_context=START
(3) Start the inference engine and load the current context rules.
(4) Perform one of the following operations depending upon current_context:
(a) Modify the statelist for correct implementation.
(b) Create blocklist, macrolist and data paths.
(c) Optimize blocklist and datapath list and perform error checks.
(d) Convert blocks to cells.
(e) Optimize cell list and perform error checks.
(f) Generate netlist.
(g) Optimize netlist and perform error checks and upon completion Goto 7.
(5) If current_context has changed, load new context rules.
(6) Goto 4.
(7) Output netlist file and stf files and Stop.

In the following sections, operations mentioned in step 4 are described. The Rule Language and PSCS display are also described.

Rule Language

The rule language of PSCS is designed to be declarative and to facilitate rule editing. In order to make the expert understand the structure of the knowledge base, the rule language provides means for knowledge representation. This will enable the format of data structures to be stated in the rule base, which will enable the expert to refer to them and understand the various structures used by the system. For example, the expert can analyze the structure of wire and determine its components. The expert can then refer these components into rules. If a new object has to be defined, then the expert can declare a new structure and modify some existing structure to link to this new structure. In this way, the growth of the data structures can be visualized better by the expert. This in turn helps the designer to update and append rules.

The following features are included in the rule language:

(i) Knowledge representation in the form of a record structure.
(ii) Conditional expressions in the antecedent of a rule.
(iii) Facility to create and destroy structure in rule actions.
(iv) The assignment statement in the action of a rule.
(v) Facility for input and output in rule actions.
(vi) Provide facility to invoke C functions from rule actions.

The rule format to be used is as follows:

The rule format to be used is as follows:

```
Rule    <number> <context>
If {
        <if-clause>
}
Then    {
        <then-clause>
}
```

where		
	<number>	rule number
	<context>	context in which this rule is active
	<if-clause>	the condition part of the rule
	<then-clause>	the action part of the rule

Inference Strategy

The inference strategy is based on a fast pattern matching algorithm. The rules are stored in a network and the requirement to iterate through the rules is avoided. This speeds up the execution. The conflict resolution strategy to be used is based on the following:

(1) The rule containing the most recent data is selected.

(2) The rule which has the most complex condition is selected.

(3) The rule declared first is selected.

Rule Editor

PSCS provides an interactive rule editor which enables the expert to update the rule set. The rules are stored in a database so that editing capabilities of the database package can be used for rule editing. To perform this operation the expert needs to be familiar with the various knowledge structures and the inferencing process. If this is not possible, then the help of a knowledge engineer is needed.

PSCS provides a menu from which various options can be set. Mechanisms are provided for setting various debugging flags and display options, and for the overall control of PSCS.

Facility is provided to save and display the blocklist created by the user. The blocklist configuration created by the user can be saved in a file and later be printed with a plotter. Also the PSCS display can be reset to restart the display process.

PSCS Example Rules:

Rule 1		
	IF	no blocks exist
	THEN	generate a system controller.
Rule 2		
	IF	a state exists which has a macro AND this macro has not been mapped to a block
	THEN	find a corresponding macro in the library and generate a block for this macro.
Rule 3		
	IF	there is a transition between two states AND there are macros in these states using the same argument
	THEN	make a connection from a register corresponding to the first macro to another register corresponding to the second macro.
Rule 4		
	IF	a register has only a single connection from another register
	THEN	combine these registers into a single register.
Rule 5		
	IF	there are two comparators AND input data widths are of the same size AND one input of these is same AND the outputs of the comparators are used to perform the same operation.
	THEN	combine these comparators into a single comparator.
Rule 6		
	IF	there is a data without a register
	THEN	allocate a register for this data.
Rule 7		
	IF	all the blocks have been interconnected AND a block has a few terminals not connected
	THEN	remove the block and its terminals, or issue an error message.
Rule 8		
	IF	memory is to be used, but a block has not been created for it
	THEN	create a memory block with data, address, read and write data and control terminals.
Rule 9		
	IF	a register has a single connection to a counter
	THEN	combine the register and the counter; remove the register and its terminals.
Rule 10		
	IF	there are connections to a terminal of a block from many different blocks
	THEN	insert a multiplexor; remove the connections to the terminals and connect them to the input of the multiplexor; connect the output of the multiplexor to the input of the block.

Additional rules address the following points:

remove cell(s) that can be replaced by using the outputs of other cell(s)

reduce multiplexor trees

use fan-out from the cells, etc.

Soft Drink Vending Machine Controller Design Example

The following example illustrates how the previously described features of the present invention are employed in the design of an application specific integrated circuit (ASIC). In this illustrative example the ASIC is designed for use as a vending machine controller. The vending machine controller receives a signal each time a coin has been deposited in a coin receiver. The coin value is recorded and when coins totalling the correct amount are received, the controller generates a signal to dispense a soft drink. When coins totalling more than the cost of the soft drink are received, the controller dispenses change in the correct amount.

This vending machine controller example is patterned after a textbook example used in teaching digital system controller design. See Fletcher, William I., *An Engineering Approach to Digital Design*, Prentice-Hall, Inc., pp. 491–505. Reference may be made to this textbook example for a more complete explanation of this vending machine controller requirements, and for an understanding and appreciation of the complex design procedures prior to the present invention for designing the hardware components for a controller.

FIG. 10 illustrates a flowchart for the vending machine controller system. This flowchart would be entered into the KBSC system by the user through the flowchart editor. Briefly reviewing the flowchart, the controller receives a coin present signal when a coin is received in the coin receiver. State0 and cond0 define a waiting state awaiting deposit of a coin. The symbol CP represents "coin present" and the symbol !CP repre-

sents "coin not present". Statel and condI determine when the coin has cleared the coin receiver. At state20, after receipt of a coin, the macro instruction ADD3.1 (lc, cv, sum) instructs the system to add lc (last coin) and cv (coin value) and store the result as sum. The macro instruction associated with state21 moves the value in the register sum to cv. The macro CMP.1 at state22 compares the value of cv with PR (price of soft drink) and returns signals EQ, GT and LT. The condition cond2 tests the result of the compare operation CMP.1. If the result is "not greater than" (!GT.CMP.1), then the condition cond3 tests to see whether the result is "equal" (EQ.CMP.1). If the result is "not equal" (!EQ.CMP.1), then control is returned to state0 awaiting the deposit of another coin. If cond3 is EQ, then state4 generates a control signal to dispense a soft drink (droppop) and the macro instruction CLR.1(cv) resets cv to zero awaiting another customer.

If the total coins deposited exceed the price, then state30 produces the action "returncoin". Additionally, the macro DECR.1 (cv) reduces the value of cv by the amount of the returned coin. At state31 cv and PR are again compared. If cv is still greater than PR, then control passes to state30 for return of another coin. The condition cond5 tests whether the result of CMP.2 is EQ and will result in either dispensing a drink (droppop) true or branching to state0 awaiting deposit of another coin. The macros associated with the states shown in FIG. 10 correspond to those defined in Table 1 above and define the particular actions which are to be performed at the respective states.

Appendix A shows the intermediate file or "statelist" produced from the flowchart of FIG. 10. This statelist is produced as output from the EDSIM program 20 and is used as input to the PSCS program 30 (FIG. 3).

FIG. 11 illustrates for each of the macros used in the flowchart of FIG. 10, the corresponding hardware blocks. It will be seen that the comparison macro CMP (A,B) results in the generation of a register for storing value A, a register for storing value B, and a comparator block and also produces control paths to the system controller for the EQ, LT, and GT signals generated as a result of the comparison operation. The addition macro ADD (A,B,C) results in the generation of a register for each of the input values A and B, a register for the output value C, and in the generation of an adder block. The macro DECR (A) results in the generation of a counter block. The PSCS program 30 maps each of the macros used in the flowchart of FIG. 10 to the corresponding hardware components results in the generation of the hardware blocks shown in FIG. 12. In generating the illustrated blocks, the PSCS program 30 relied upon rules 1 and 2 of the above listed example rules.

FIG. 13 illustrates the interconnection of the block of FIG. 12 with data paths and control paths. Rule 3 was used by the data/control path synthesizer program 31 in mapping the data and control paths.

FIG. 14 shows the result of optimizing the circuit by applying rule 4 to eliminate redundant registers. As a result of application of this rule, the registers R2, R3, R7, R8, and R9 in FIG. 13 were removed. FIG. 15 shows the block diagram after further optimization in which redundant comparators are consolidated. This optimization is achieved in the PSCS program 30 by application of rule 5.

Having now defined the system controller block, the other necessary hardware blocks and the data and control paths for the integrated circuit, the PSCS program 30 now generates a netlist 15 defining these hardware components and their interconnection requirements. From this netlist the mask data for producing the integrated circuit can be directly produced using available VLSI CAD tools.

```
name rpop;
data path @ic<0:5>, cv<0:5>, sum<0:5>, @pr<0:5>;
{
state4 : state0;
state30 : state31;
state21 : state22;
state20 : state21;
state0 :. !cp state0;
state0 :. cp state1;
state1 :. cp state1;
state1 :. !cp state20;
state22 :. GT.CMP.1 state30;
state22 :. !GT.CMP.1*EQ.CMP.1 state4;
state22 :. !GT.CMP.1*!EQ.CMP.1 state0;
state31 :. GT.CMP.2 state30;
state31 :. !GT.CMP.2*EQ.CMP.2 state4;
state31 :. !GT.CMP.2*!EQ.CMP.2 state0;
state30 :: returncoin;
state30 :: DECR.1(cv);
state4 :: droppop;
state4 :: CLR.1(cv);
state31 :: CMP.2(cv,pr);
state22 :: CMP.1(cv,pr);
state21 :: MOVE.1(sum,cv);
state20 :: ADD3.1(ic,cv,sum);
}
```

That which I claimed is:

1. A computer-aided design system for designing an application specific integrated circuit directly from architecture independent functional specifications for the integrated circuit, comprising

a macro library defining a set of architecture independent operations comprised of actions and conditions;

input specification means operable by a user for defining architecture independent functional specifications for the integrated circuit, said functional specifications being comprised of a series of operations comprised of actions and conditions, said input specification means including means to permit the user to specify for each operation a macro selected from said macro library;

a cell library defining a set of available integrated circuit hardware cells for performing the available operations defined in said macro library;

cell selection means for selecting from said cell library for each macro specified by said input specification means, appropriate hardware cells for performing the operation defined by the specified macro, said cell selection means comprising an expert system including a knowledge base containing rules for. selecting hardware cells from said cell library and inference engine means for selecting appropriate hardware cells from said cell library in accordance with the rules of said knowledge base; and

netlist generator means cooperating with said cell selection means for generating as output from the system a netlist defining the hardware cells which are needed to achieve the functional requirements of the integrated circuit and the connections therebetween.

2. The system as defined in claim 1 wherein said input means comprises means specification for receiving user

input of a list defining the series of actions and conditions.

3. The system as defined in claim 1 additionally including mask data generator means for generating from said netlist the mask data required to produce an integrated circuit having the specified functional requirements.

4. The system as defined in claim 1 wherein said input means comprises flowchart editor means specification for creating a flowchart having elements representing said series of actions and conditions.

5. The system as defined in claim 4 additionally including flowchart simulator means for simulating the functions defined in the flowchart to enable the user to verify the operation of the integrated circuit.

6. The system as defined in claim 1 additionally including data path generator means cooperating with said cell selection means for generating data paths for the hardware cells selected by said cell selection means.

7. The system as defined in claim 6 wherein said data path generator means comprises a knowledge base containing rules for selecting data paths between hardware cells and inference engine means for selecting data paths between the hardware cells selected by said cell selection means in accordance with the rules of said knowledge base and the arguments of the specified macros.

8. The system as defined in claim 6 additionally including control generator means for generating a controller and control paths for the hardware cells selected by said cell selection means.

9. A computer-aided design system for designing an application specific integrated circuit directly from a flowchart defining architecture independent functional requirements of the integrated circuit comprising

a marco library defining a set of architecture independent operations comprised of actions and conditions;

flowchart editor means operable by a user for creating a flowchart having elements representing said architecture independent operations;

said flowchart editor means including macro specification means for permitting the user to specify for each operation represented in the flowchart a macro selected from said macro library;

a cell library defining a set of available integrated circuit hardware cells for performing the available operations defined in said macro library;

cell selection means for selecting form said cell library for each specified macro, appropriate hardware cells for performing the operation defined by the specified macro, said cell selection means comprising an expert system including a knowledge base containing rules for selecting hardware cells from said cell library and inference engine means for selecting appropriate hardware cells from said cell library in accordance with the rules of said knowledge base; and

data path generator means cooperating with said cell selection means for generating data paths for the hardware cells selected by said cell selector means, said data path generator means comprising a knowledge base containing rules for selecting data paths between hardware cells and inference engine means for selecting data paths between hardware cells selected by said cell selection means in accordance with the rules of said knowledge base and the arguments of the specified macros.

10. The system as defined in claim 9 additionally including control generator means for generating a controller and control paths for the hardware cells selected by said cell selection means.

11. A computer-aided design system for designing an application specific integrated circuit directly from a flowchart defining architecture independent functional requirements of the integrated circuit, comprising

flowchart editor means operable by a user for creating a flowchart having boxes representing architecture independent actions, diamonds representing architecture independent conditions, and lines with arrows representing transitions between actions and condition and including means for specifying for each box or diamond, a particular action or condition to be performed;

a cell library defining a set of available integrated circuit hardware cells for performing actions and conditions;

a knowledge base containing rules for selecting hardware cells from said cell library and for generating data and control paths for hardware cells; and

expert system means operable with said knowledge base for translating the flowchart defined by said flowchart editor means into a netlist defining the necessary hardware cells and data and control paths required in an integrated circuit having the specified functional requirements.

12. The system as defined in claim 11 including mask data generator means for generating from said netlist the mask data required to produce an integrated circuit having the specified functional requirements.

13. A computer-aided design process for designing an application specific integrated circuit which will perform a desired function comprising

storing a set of definitions of architecture independent actions and conditions;

storing data describing a set of available integrated circuit hardware cells for performing the actions and conditions defined in the stored set;

storing in an expert system knowledge base a set of rules for selecting hardware cells to perform the actions and conditions;

describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions;

specifying for each described action and condition of the series one of said stored definitions which corresponds to the desired action or condition to be performed; and

selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit, said step of selecting a hardware cell comprising applying to the specified definition of the action or condition to be performed, a set of cell selection rules stored in said expert system knowledge base and generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor.

14. A process as defined in claim 13, including generating from the netlist the mask data required to produce an integrated circuit having the desired function.

15. A process as defined in claim 13 including the further step of generating data paths for the selected integrated circuit hardware cells.

16. A process as defined in claim 15 wherein said step of generating data paths comprises applying to the selected cells a set of data path rules stored in a knowledge base and generating the data paths therefrom.

17. A process as defined in claim 16 including the further step of generating control paths for the selected integrated circuit hardware cells.

18. A knowledge based design process for designing an application specific integrated circuit which will perform a desired function comprising

storing in a macro library a set of macros defining architecture independent actions and conditions;

storing in a cell library a set of available integrated circuit hardware cells for performing the actions and conditions;

storing in a knowledge base set of rules for selecting hardware cells from said cell library to perform the actions and conditions defined by the stored macros;

describing for a proposed application specific integrated circuit a flowchart comprised of elements representing a series of architecture independent actions and conditions which carry out the function to be performed by the integrated circuit;

specifying for each described action and condition of said series a macro selected from the macro library which corresponds to the action or condition; and

applying rules of said knowledge base to the specified macros to select from said cell library the hardware cells required for performing the desired function of the application specific integrated circuit and generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor.

19. A process as defined in claim 18 also including the steps of

storing in said knowledge base a set of rules for creating data paths between hardware cells, and

applying rules of said knowledge base to the specified means to create data paths for the selected hardware cells.

20. A process as defined in claim 19 also including the steps of generating a controller and generating control paths for the selected hardware cells.

* * * * *

EXHIBIT B

US005197016A

United States Patent [19]

Sugimoto et al.

[11] Patent Number: 5,197,016

[45] Date of Patent: Mar. 23, 1993

[54] **INTEGRATED SILICON-SOFTWARE COMPILER**

[75] Inventors: **Tai Sugimoto; Hideaki Kobayashi,** both of Columbia, S.C.; **Masahiro Shindo; Haruo Nakayama,** both of Osaka, Japan

[73] Assignees: **International Chip Corporation,** Columbia, S.C.; **Ricoh Company, Ltd.,** Tokyo, Japan

[21] Appl. No.: **380,079**

[22] Filed: **Jul. 14, 1989**

Related U.S. Application Data

[63] Continuation-in-part of Ser. No. 143,821, Jan. 13, 1988, Pat. No. 4,922,432.

[51] Int. Cl.[5] **G06F 15/60**

[52] U.S. Cl. **364/490**; 364/489; 364/488

[58] Field of Search 364/488, 489, 490, 491

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,648,044	3/1987	Hardy et al.	395/76
4,658,370	4/1987	Erman et al.	395/76
4,675,829	6/1987	Clemenson	395/65
4,703,435	10/1987	Darringer et al.	364/489
4,922,432	5/1990	Kobayashi et al.	364/490

OTHER PUBLICATIONS

"A Front End Graphic Interface to the First Silicon Compiler" by J. H. Nash et al., European Conf. on Electronic Design Automation (EDA84), pp. 120–124.

An Engineering Approach to Digital Design, William I. Fletcher, Prentice-Hall, Inc. pp. 491–505.

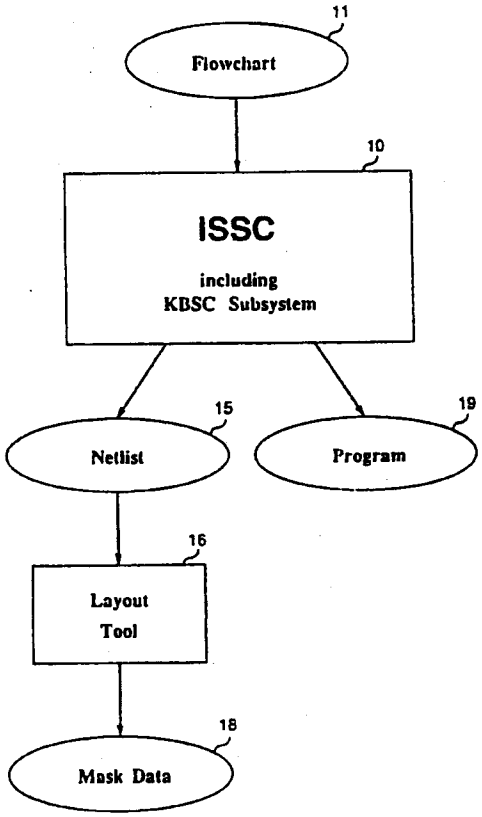
Primary Examiner—Vincent N. Trans

Attorney, Agent, or Firm—Bell, Seltzer, Park & Gibson

[57] **ABSTRACT**

A computer-aided system and method is disclosed for designing an application specific integrated circuit (ASIC) whose intended function is implemented both by a hardware subsystem including hardware elements on the integrated circuit and by a software subsystem including a general purpose microprocessor also on the integrated circuit. The system also generates software instructions for use by the software subsystem. The system utilizes a knowledge based expert system, with a knowledge base extracted from expert ASIC designers, and thus makes it possible for ASIC's to be designed and provided quickly and economically by persons not having the highly specialized skill of an ASIC designer.

32 Claims, 7 Drawing Sheets



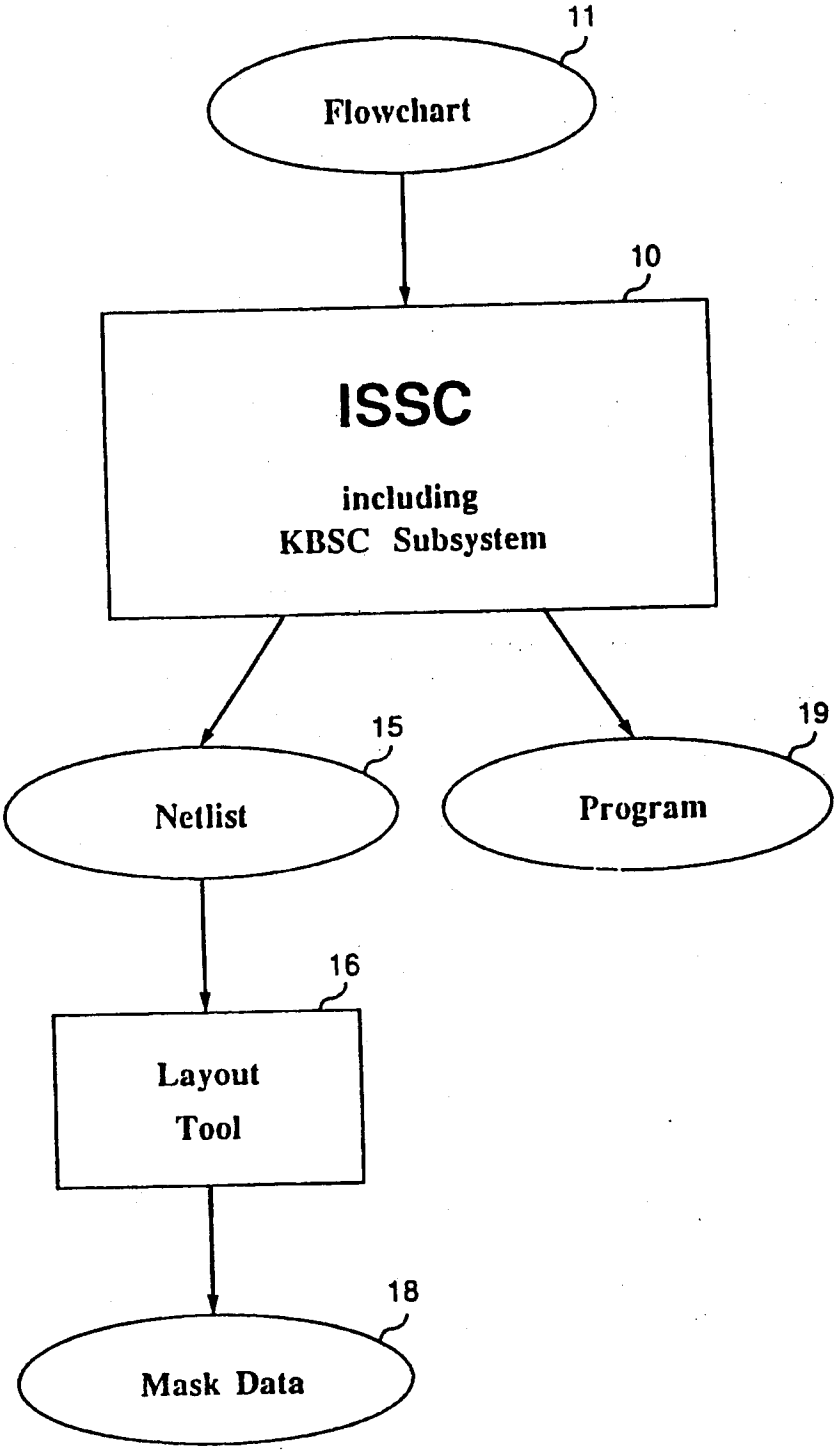

11
Flowchart
10
ISSC
including
KBSC Subsystem
15
Netlist
19
Program
16
Layout
Tool
18
Mask Data

FIG. 1.

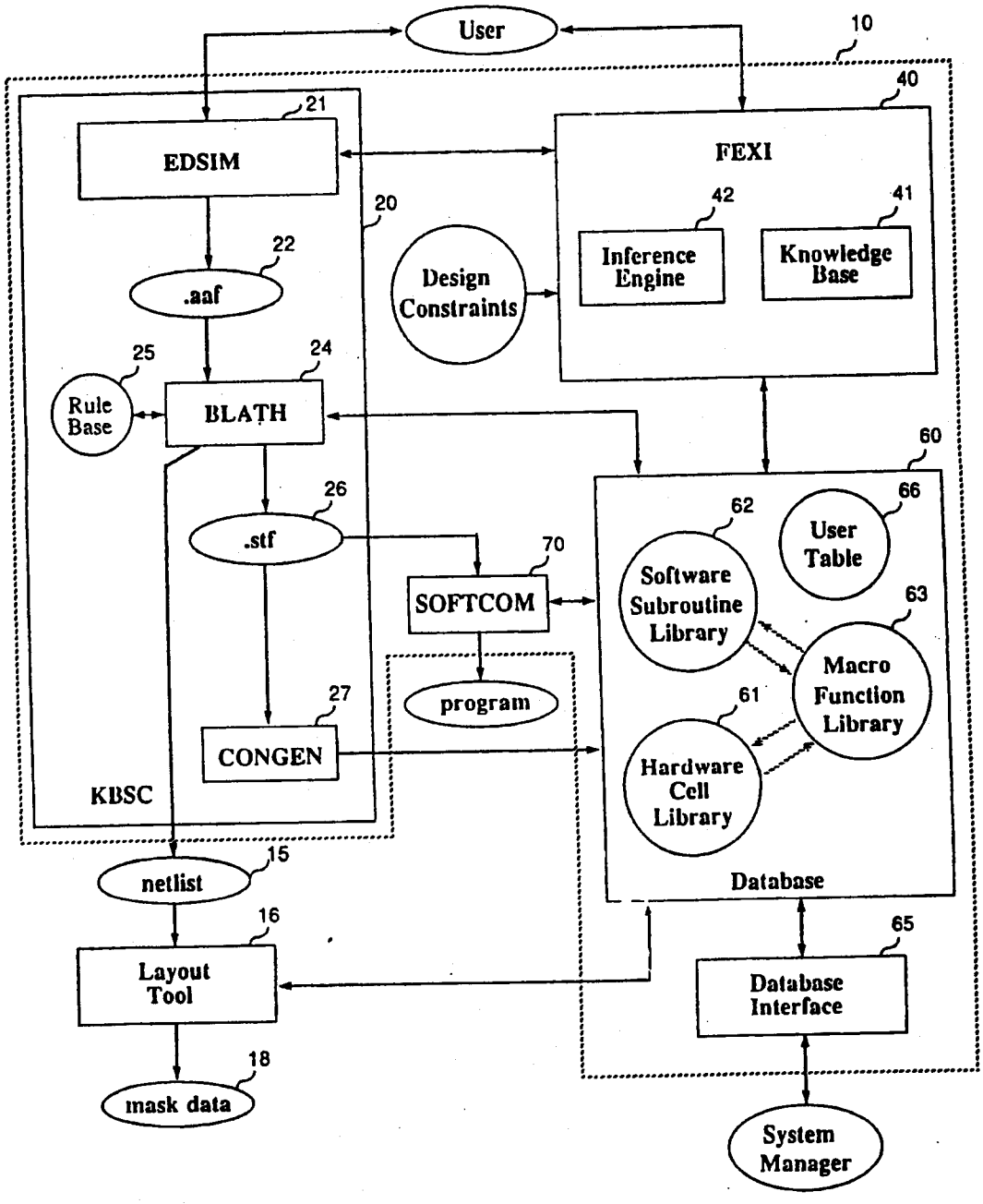

User
10
40
21
EDSIM
FEXI
42
41
20
Inference Engine
Knowledge Base
Design Constraints
22
.aaf
25
24
Rule Base
BLATH
60
26
.stf
62
66
User Table
70
SOFTCOM
Software Subroutine Library
63
Macro Function Library
program
61
27
CONGEN
Hardware Cell Library
KBSC
15
netlist
Database
16
65
Layout Tool
Database Interface
18
mask data
System Manager

FIG. 2.

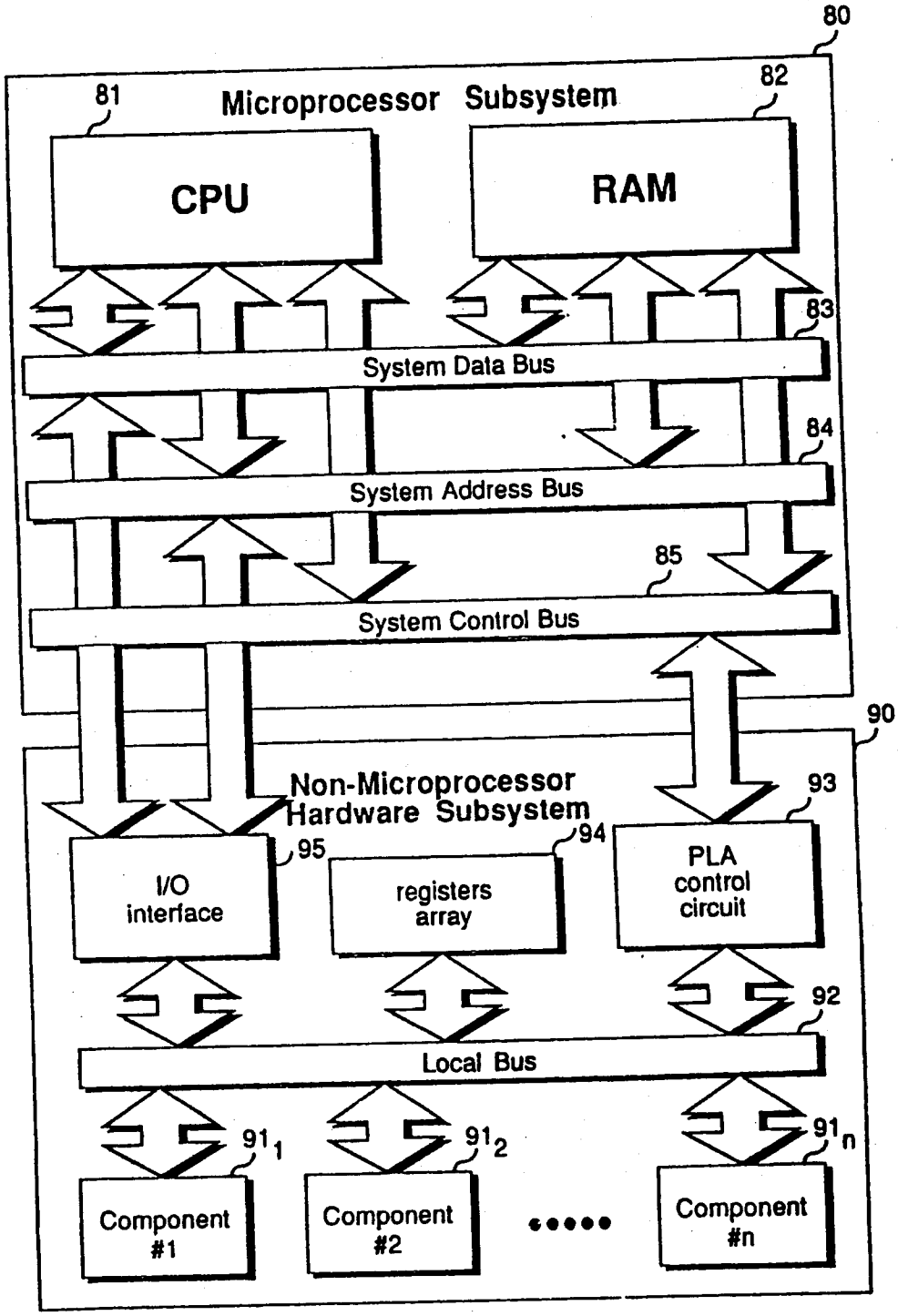

80
81
Microprocessor Subsystem
82
CPU
RAM
83
System Data Bus
84
System Address Bus
85
System Control Bus
90
Non-Microprocessor
Hardware Subsystem
93
95
94
I/O
interface
registers
array
PLA
control
circuit
92
Local Bus
91 1
91 2
91 n
Component
#1
Component
#2
Component
#n

FIG. 3.

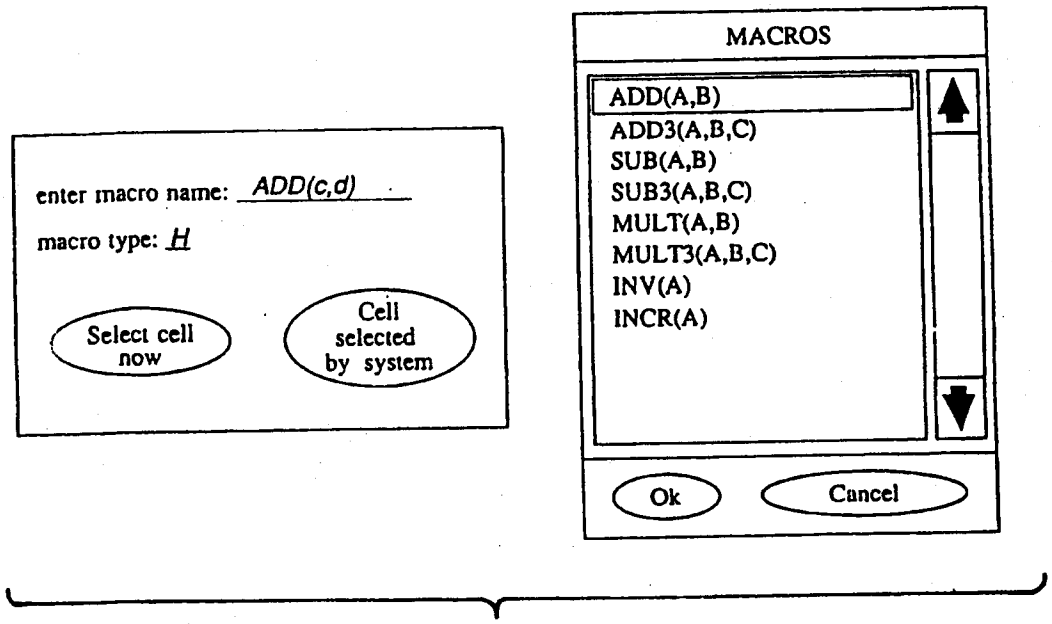

enter macro name: ADD(c,d)
macro type: H
Select cell now
Cell selected by system
MACROS
ADD(A,B)
ADD3(A,B,C)
SUB(A,B)
SUB3(A,B,C)
MULT(A,B)
MULT3(A,B,C)
INV(A)
INCR(A)
Ok
Cancel

FIG. 4.

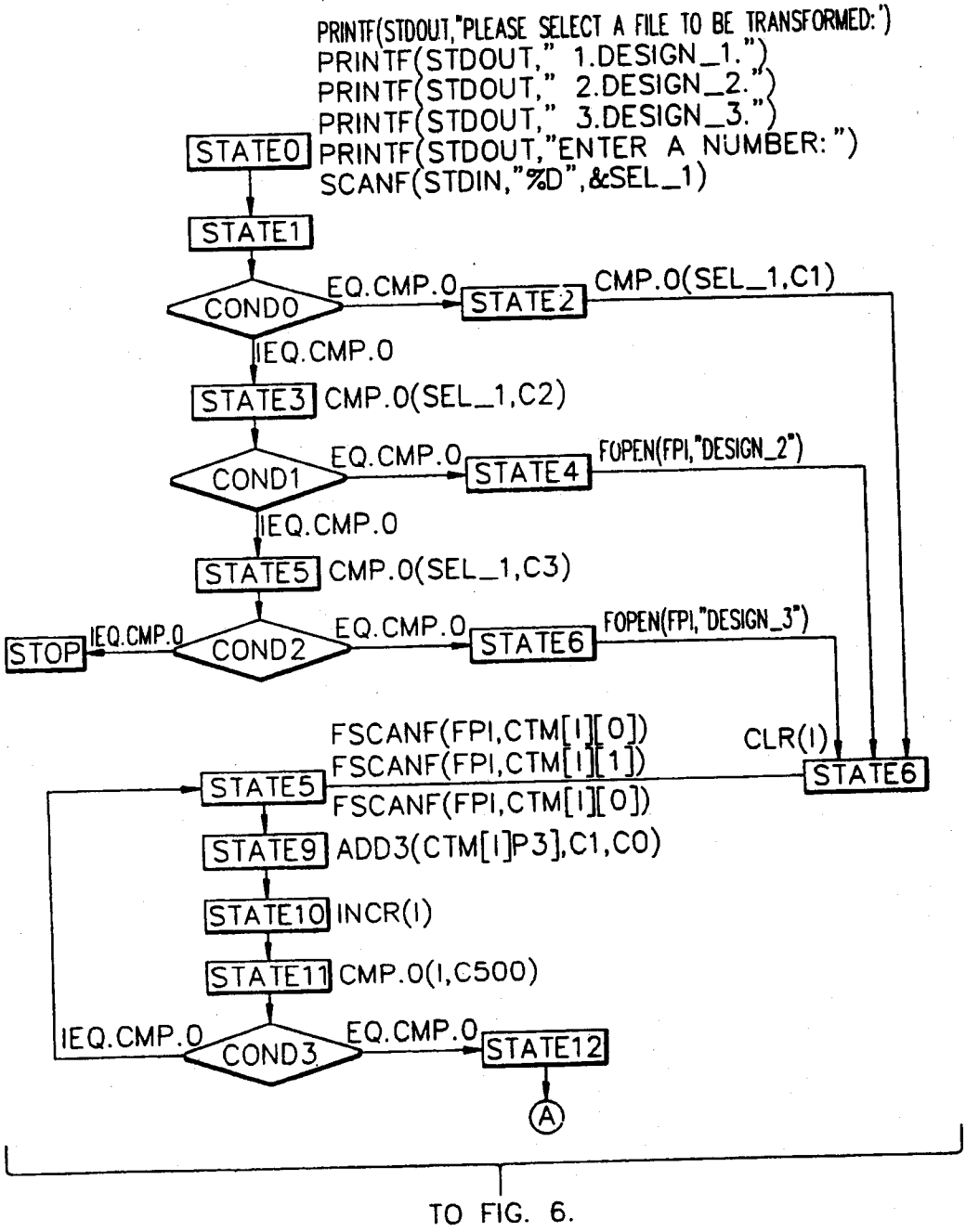

PRINTF(STDOUT,"PLEASE SELECT A FILE TO BE TRANSFORMED:")
PRINTF(STDOUT," 1.DESIGN_1.")
PRINTF(STDOUT," 2.DESIGN_2.")
PRINTF(STDOUT," 3.DESIGN_3.")
PRINTF(STDOUT,"ENTER A NUMBER:")
SCANF(STDIN,"%D",&SEL_1)
STATE0
STATE1
COND0
EQ.CMP.0
STATE2
CMP.0(SEL_1,C1)
IEQ.CMP.0
STATE3
CMP.0(SEL_1,C2)
COND1
EQ.CMP.0
STATE4
FOPEN(FPI,"DESIGN_2")
IEQ.CMP.0
STATE5
CMP.0(SEL_1,C3)
STOP
IEQ.CMP.0
COND2
EQ.CMP.0
STATE6
FOPEN(FPI,"DESIGN_3")
FSCANF(FPI,CTM[I][0])
FSCANF(FPI,CTM[I][1])
FSCANF(FPI,CTM[I][0])
CLR(I)
STATE5
STATE6
STATE9
ADD3(CTM[I]P3],C1,C0)
STATE10
INCR(I)
STATE11
CMP.0(I,C500)
IEQ.CMP.0
COND3
EQ.CMP.0
STATE12
A
TO FIG. 6.

FIG. 5.

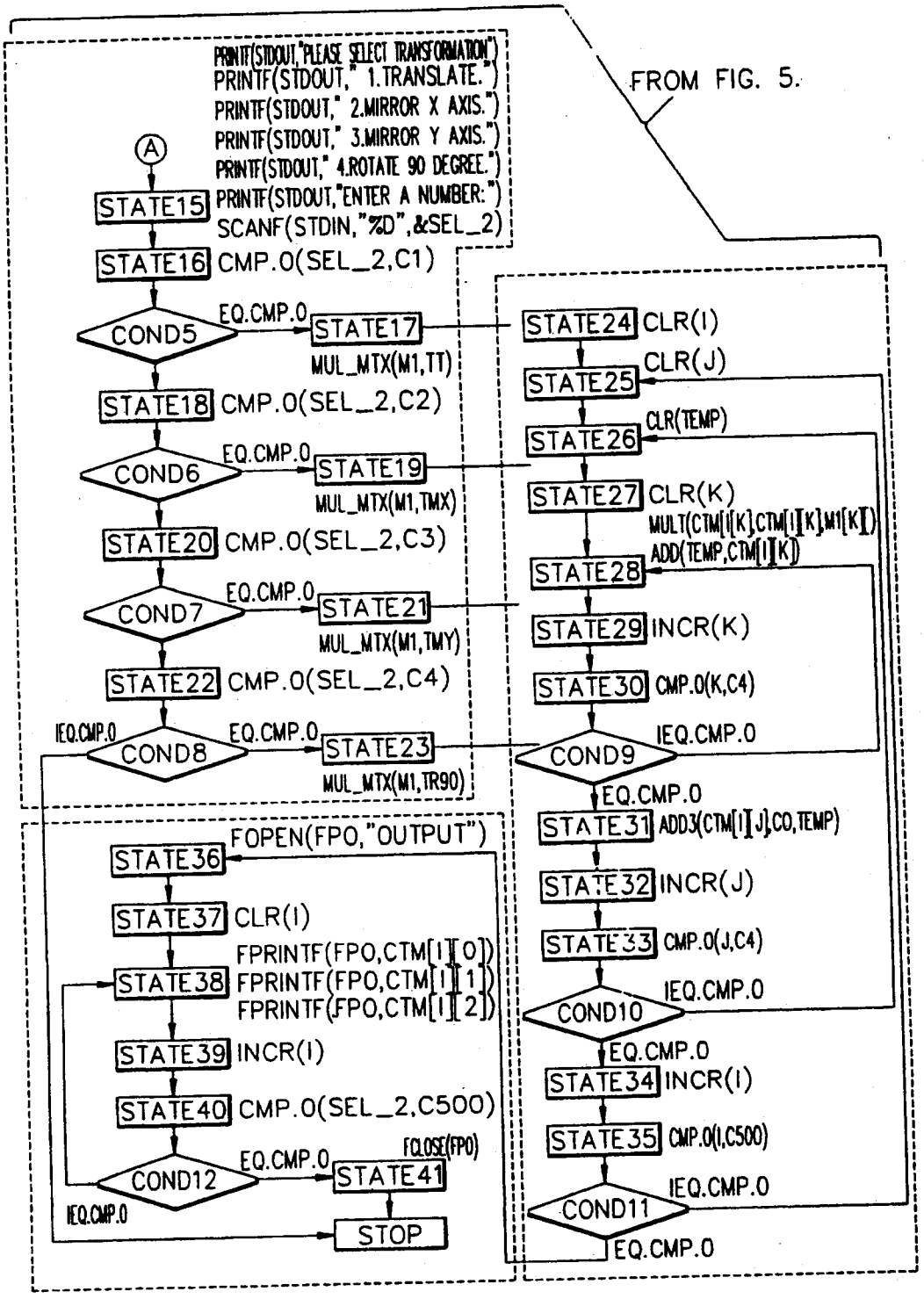

FROM FIG. 5.
PRINTF(STDOUT,"PLEASE SELECT TRANSFORMATION")
PRINTF(STDOUT," 1.TRANSLATE.")
PRINTF(STDOUT," 2.MIRROR X AXIS.")
PRINTF(STDOUT," 3.MIRROR Y AXIS.")
PRINTF(STDOUT," 4.ROTATE 90 DEGREE.")
A
STATE15 PRINTF(STDOUT,"ENTER A NUMBER:")
SCANF(STDIN,"%D",&SEL_2)
STATE16 CMP.0(SEL_2,C1)
COND5 EQ.CMP.0 STATE17 MUL_MTX(M1,TT)
STATE18 CMP.0(SEL_2,C2)
COND6 EQ.CMP.0 STATE19 MUL_MTX(M1,TMX)
STATE20 CMP.0(SEL_2,C3)
COND7 EQ.CMP.0 STATE21 MUL_MTX(M1,TMY)
STATE22 CMP.0(SEL_2,C4)
COND8 IEQ.CMP.0 EQ.CMP.0 STATE23 MUL_MTX(M1,TR90)
STATE24 CLR(I)
STATE25 CLR(J)
STATE26 CLR(TEMP)
STATE27 CLR(K)
STATE28 MULT(CTM[I][K],CTM[I][K],M1[K][I])
ADD(TEMP,CTM[I][K])
STATE29 INCR(K)
STATE30 CMP.0(K,C4)
COND9 IEQ.CMP.0 EQ.CMP.0
STATE31 ADD3(CTM[I][J],C0,TEMP)
STATE32 INCR(J)
STATE33 CMP.0(J,C4)
COND10 IEQ.CMP.0 EQ.CMP.0
STATE34 INCR(I)
STATE35 CMP.0(I,C500)
COND11 IEQ.CMP.0 EQ.CMP.0
FOPEN(FPO,"OUTPUT")
STATE36
STATE37 CLR(I)
STATE38 FPRINTF(FPO,CTM[I][0])
FPRINTF(FPO,CTM[I][1])
FPRINTF(FPO,CTM[I][2])
STATE39 INCR(I)
STATE40 CMP.0(SEL_2,C500)
COND12 EQ.CMP.0 FCLOSE(FPO) STATE41
IEQ.CMP.0
STOP

FIG. 6.

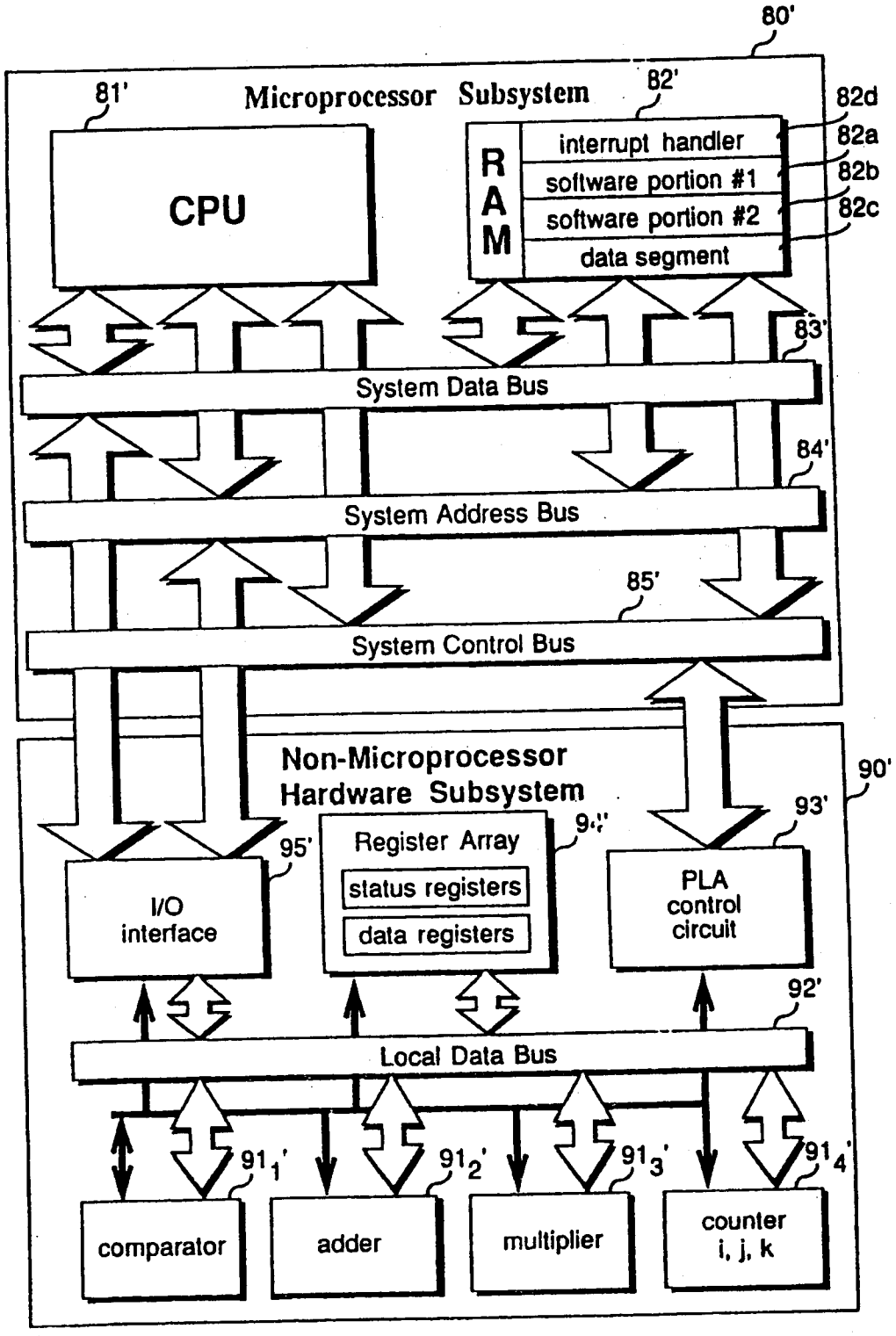

80'
81'
Microprocessor Subsystem
82'
82d
82a
82b
82c
CPU
RAM
interrupt handler
software portion #1
software portion #2
data segment
83'
System Data Bus
84'
System Address Bus
85'
System Control Bus
Non-Microprocessor
Hardware Subsystem
90'
93'
95'
Register Array
status registers
data registers
I/O
interface
PLA
control
circuit
92'
Local Data Bus
91₁'
91₂'
91₃'
91₄'
comparator
adder
multiplier
counter
i, j, k

FIG. 7.

INTEGRATED SILICON-SOFTWARE COMPILER

CROSS-REFERENCE TO RELATED APPLICATION

This application is a continuation-in-part of copending U.S. application Ser. No. 143,821, filed Jan. 13, 1988, now U.S. Pat. No. 4,922,432.

FIELD AND BACKGROUND OF THE INVENTION

This invention relates to the design of integrated circuits, and more particularly relates to a computer-aided system and method for designing application specific integrated circuits.

An application specific integrated circuit (ASIC) is an integrated circuit chip designed to perform a specific function, as distinguished from standard, general purpose integrated circuit chips, such as microprocessors, memory chips, etc. A highly skilled design engineer having specialized knowledge in VLSI circuit design is ordinarily required to design an ASIC. In the design process, the VLSI design engineer will consider the particular objectives to be accomplished and tasks to be performed by the integrated circuit and will create structural level design specifications which define the various hardware components required to perform the desired function, as well as the interconnection requirements between these components. A system controller must also be designed for synchronizing the operations of these components. This requires an extensive and all encompassing knowledge of the various hardware components required to achieve the desired objectives, as well as their interconnection requirements, signal level compatibility, timing compatibility, physical layout, etc. At each design step, the designer must do tedious analysis. The design specifications created by the VLSI design engineer may, for example, be in the form of circuit schematics, parameters or specialized hardware description languages (HDLs).

From the structural level design specifications, the description of the hardware components and interconnections is converted to a physical chip layout level description which describes the actual topological characteristics of the integrated circuit chip. This physical chip layout level description provides the mask data needed for fabricating the chip.

Due to the tremendous advances in very large scale integration (VLSI) technology, highly complex circuit systems are being built on a single chip. With their complexity and the demand to design custom chips at a faster rate, in large quantities, and for an ever increasing number of specific applications, computer-aided design (CAD) techniques need to be used. CAD techniques have been used with success in design and verification of integrated circuits, at both the structural level and at the physical layout level. For example, CAD systems have been developed for assisting in converting VLSI structural level descriptions of integrated circuits into the physical layout level topological mask data required for actually producing the chip. Although the presently available computer-aided design systems greatly facilitate the design process, the current practice still requires highly skilled VLSI design engineers to create the necessary structural level hardware descriptions.

Only a small number of VLSI designers possess the highly specialized skills needed to create structural level integrated circuit hardware descriptions. Even

with the assistance of available VLSI CAD tools, the design process is time consuming and the probability of error is also high because of human involvements. There is a very significant need for a better and more cost effective way to design custom integrated circuits.

In U.S. application Ser. No. 143,821, filed Jan. 13, 1988, and entitled *Knowledge Based Method and Apparatus for Designing Integrated Circuits Using Functional Specifications*, there is disclosed a computer-aided design system and method which enables a user to define the functional requirements for a desired application specific integrated circuit using an easily understood architecture independent functional level representation, such as a flowchart. From this functional level description, a computer implemented expert system generates the detailed structural level definitions needed for producing the application specific integrated circuit. The structural level definitions include a list of the integrated circuit hardware cells needed to achieve the functional specifications. Also included in the detailed structural definitions are the data paths among the selected hardware cells, a system controller for coordinating the operation of the cells and control paths for the selected integrated circuit cells. The various hardware cells are selected from a cell library of previously designed hardware cells of various functions and technical specifications. From this detailed structural level definition it is possible, using either known manual techniques or existing VLSI CAD layout systems to generate the detailed chip level geometrical information (e.g. mask data) required to produce the particular application specific integrated circuit in chip form.

The system described in the aforementioned copending application provides a very significant advance over the methods previously available for designing application specific integrated circuits and opens the possibility for the design and production of application specific integrated circuits by designers, engineers and technicians who may not possess the specialized expert knowledge of a highly skilled VLSI design engineer.

SUMMARY OF THE INVENTION

The present invention provides an improvement over and an extension to the system and method of the aforementioned copending application. Although the system and method of the aforementioned application provides an excellent means for designing ASICs whose functions are implemented in hardware form, there are some occasions where particular functions of the ASCI would best be implemented by software.

The present invention provides a computer-aided system and method for designing an application specific integrated circuit whose intended function is implemented by both a hardware subsystem including hardware elements on the integrated circuit and by a software subsystem including a general purpose microprocessor. Thus, the resulting ASIC produced by the computer-aided design system and method of the present invention includes, on a single integrated circuit chip, a microprocessor for executing the software instructions of the software subsystem and various integrated circuit hardware cells for performing other functions of the integrated circuit.

The system more particularly utilizes a knowledge based expert system, with a knowledge base extracted from expert ASIC designers with a high level of expertise in system design. The knowledge base contains

rules for selecting software subroutines from a software subroutine library of predefined functions or operations, as well as rules for selecting hardware cells from a cell library of predefined cells of various types and functions. An inference engine is provided for selecting appropriate hardware cells or software subroutines from these libraries in accordance with the rules of the knowledge base.

The functional specifications of the desired ASIC are independent of any particular architecture or design style and can be defined in a suitable manner, such as in text form or preferably in a flowchart format. The flowchart is a highly effective means of describing a sequence of logical operations, and is well understood by software and hardware designers of varying levels of expertise and training. The designer, when defining the series of operations which implement the intended function of the application specific integrated circuit, may specify whether a particular operation should be implemented in hardware or in software.

The designer may also specify design constraints for the ASIC, such as annual volume, speed, size, pin count, packaging type, power consumption and thermal stability. The knowledge base contains design style rules for selecting, based upon the specified design constraints, an optimum design style for implementation of the hardware subsystem employing various technologies such as programmable logic device (PLD), gate array, standard cell and macro cell.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be better understood by reference to the detailed description which follows, taken in connection with the accompanying drawings, in which

FIG. 1 is a block schematic diagram showing how integrated circuit mask data is created from flowchart descriptions by the system of the present invention;

FIG. 2 is a block schematic diagram showing the various elements of the integrated silicon-software compiler system and method of the present invention;

FIG. 3 is a schematic illustration of the system configuration of an ASIC containing both a microprocessor subsystem and a non-microprocessor hardware subsystem;

FIG. 4 is an illustration of a computer display screen which the designer uses in defining the functional specifications of the integrated circuit;

FIGS. 5 and 6 are flowcharts illustrating the functions to be performed by an ASIC in an illustrative design example of the system and method of the invention; and

FIG. 7 is a schematic illustration similar to FIG. 3, but showing the particular ASIC of the design example.

MORE DETAILED DESCRIPTION OF THE INVENTION

System Overview

The overall system flow in accordance with the present invention is illustrated in FIG. 1. The computer-aided system and method of the present invention is represented by the block **10**, and for simplicity is referred to herein by the acronym ISSC (integrated silicon-software compiler). The ISSC **10** receives as its input (which are technology and architecture independent) specifications **11**, and which define on a functional or behavioral level the functions which are to be performed by the target application specific integrated circuit (ASIC). As illustrated in FIG. **1**, the user enters the functional specifications of the desired target ASIC into the integrated silicon-software compiler (ISSC) **10** in the form of a flowchart **11**. The ISSC **10** then generates a netlist **15** from the flowchart. The netlist **15** includes architecture specific definitions of the hardware cells required to implement the functions of the hardware subsystem, a definition of the microprocessor used to implement the functions of the software subsystem, and definitions of a system controller, data paths and control paths for interconnecting the various components. The netlist **15** can be used as input to any existing VLSI layout and routing tool **16** to create mask data **18** for geometrical layout. The ISSC system **10** also generates a program **19** from the flowchart information for implementing the functions of the software subsystem. The program **19** can be stored in an external or internal memory associated with the microprocessor on the target ASIC.

The ISSC system **10** can be operated on a suitable programmed general purpose digital computer. By way of example, one embodiment of the system is operated in a workstation environment such as Sun3 and VAX-Station-II/GPX running UNIX Operating System and X Window Manager. The software uses C programming language and a data base such as INGRES or Gbase. The human interface is mainly done by the use of a pointing device, such as a mouse, with pop up menus, buttons, and a special purpose command language. The permanent data of the integrated circuit design are stored in a database for easy retrieval and update. Main memory temporarily stores the executable code, design data (flowchart, logic, etc.), data base (cell library), and knowledge base. The CPU performs the main tasks of creating and simulating flowcharts and the automatic synthesis of the design.

The primary elements or modules which comprise the ISSC system are shown in FIG. **2**. In the embodiment illustrated and described herein, these elements or modules are in the form of software programs, although persons skilled in the appropriate art will recognize that these elements can be easily embodied in other forms, such as in hardware for example.

Referring more particularly to FIG. **2**, it will be seen that the ISSC system **10** includes a module or subsystem **20** called KBSC (Knowledge Based Silicon Compiler). The KBSC subsystem **20** corresponds to the system and method described in copending U.S. application Ser. No. 143,821, filed Jan. 13, 1988, the subject matter of which is incorporated herein by reference. As shown in FIG. **2**, the KBSC subsystem **20** includes a number of modules or programs which collectively provide an interface with the user for receiving input of the functional specifications for the particular target ASIC and which provide as output a netlist **15** and a program **19**. Other major subsystems or modules of the ISSC **10** include a front-end expert system interface (FEXI) **40**, a database **60** and a software compiler (SOFTCOM) **70**.

The KBSC subsystem **20** includes a program **21** called EDSIM (EDitor SIMulator), which comprises a flowchart editor for creating and editing flowcharts and a flowchart simulator for simulation and verification of flowcharts. The output of EDSIM **21** is an intermediate file **22**, referred to as an Antecedent-Action-Form (AAF), which contains a behavioral description of the system that is to be designed. The AAF file **22** contains information about storage elements which compose the data paths, memory elements and external connections

to the system. The AAF file **22** is the input to the BLATH program **24**.

The BLATH (Block Level Aaf To Hardware) program **24** is a knowledge based logic synthesis program which selects optimum hardware cells for the hardware subsystem from the hardware cell library and also selects a microprocessor or CPU megacell for use by the software subsystem. The selection is based upon functional descriptions in the flowchart, as specified by the macros assigned to each action represented in the flowchart. BLATH **24** uses a knowledge base **25** extracted from VLSI design experts to select the hardware cells and microprocessor based upon design constraint rules, design style selection rules, and software/hardware routine selection rules. BLATH **24** selects from a database **60** which includes a hardware cell library **61** of previously designed hardware cells and a software subroutine library **62** of previously designed software subroutines. BLATH **24** also outputs an STF file **26** which is used by a system controller generator CONGEN (CONtroller GENerator) **27** and by the software compiler (SOFTCOM) **70**. The controller generator **27** generates a custom designed system controller for controlling the operation of the hardware cells and coordinating with the microprocessor of the software subsystem. Thus, with a functional flowchart input from EDSIM **21**, BLATH **24** selects a microprocessor and all required hardware cells, generates data and control paths, and generates a netlist **15** describing all of ASIC design information.

Target ASIC Design

FIG. **3** illustrates the system configuration of an application specific integrated circuit (ASIC) designed using the ISSC **10**. It consists of two subsystems, a microprocessor subsystem **80** and a non-microprocessor hardware subsystem **90**. The microprocessor subsystem contains a general purpose microprocessor CPU **81** and random access memory (RAM) **82** to execute the software functions of the ASIC. The software program is stored in RAM memory **82**. The hardware subsystem **90** contains special purpose hardware components or cells which perform the functions of the ASIC. Functions are executed in special hardware components primarily to reduce processing time. The various hardware components 91_1, 91_2 . . . 91_n are integrated into a subsystem which includes a local bus **92**, a PLA control circuit **93**, a register array **94** and an input/output interface **95**. These two subsystems communicate with each other through a system data bus **83**, a system address bus **84** and a system control bus **85**.

EDSIM

The creation and verification of the flowchart is the first step in the VLSI design methodology. The translation from an algorithm to an equivalent flowchart is performed with the flowchart editor, which is contained within the program EDSIM **21**. The flowchart editor provides a working environment for interactive flowchart editing with a designer friendly interface. A graphical display of the flowchart is provided consisting of boxes, diamonds, and lines. All are drawn on the screen and look like a traditional flowchart. The flowchart editor also provides functions such as loading and saving flowcharts. EDSIM will generate an intermediate file **22** for each flowchart. This file is then used by the BLATH program **24** to generate a netlist **15**.

The main editing functions of the flowchart editor include, create, edit, and delete states, conditions, and transitions. The create operation allows the designer to add a new state, condition, or transition to a flowchart. Edit allows the designer to change the position of a state, condition or transition, and delete allows the designer to remove a state, condition or transition from the current flowchart. States which contain actions are represented by boxes, conditions are represented by diamonds, and transitions are represented by lines with arrows showing the direction of the transition.

Once the states and transitions have been created, the designer assigns operations to each state. These operations are made up of macro functions and arguments. The macro function library **63** in database **60** contains a set of macros defining various operations corresponding to the available actions and conditions which can be specified in a flowchart. During the operation of the EDSIM program **21**, the user assigns to each block in the flowchart a macro selected from the macro library and the associated parameters of the macro, if any. FIG. **4** illustrates a screen provided for the designer to specify a macro function in a state in the flowchart. The designer types a macro name along with its parameters in the field associated with the label "enter macro name". In the righthand portion of the screen a macro list is displayed for the designer to select a macro, which can be selected by pointing and clicking a mouse. Once a macro has been selected by this method, the designer would then specify the parameters of the macro. The designer can also select the desired macro type, e.g. hardware (H) or software (S). He then clicks on one of the two circles, "select cell now" or "cell selected by system". If "select cell now" is selected, then EDSIM **21** creates and sends a query to FEXI **40**. The expert system FEXI then queries the database and retrieves suitable cells (either hardware or software as desired by the user). If neither is selected by the user, then both types of cells are displayed. FEXI displays this cell list to the designer and waits for him to select a cell. The designer can query about the features of the cells. EDSIM then returns to normal flowchart editing. If the designer had selected "cell selected by system" from the above screen, then FEXI is not invoked and normal flowchart editing continues.

A list of basic macro functions available in the macro function library **63** is shown in Table 1.

TABLE 1

id	macro name	parameters	description
1	ADD	2	B = A + B
2	ADD3	3	C = A + B
3	SUB	2	B = A − B
4	MULT	3	C = A * B
5	DIV	3	C = A/B
6	DEC	1	A = A − 1
7	INCR	1	A = A + 1
8	REG	2	B = A
9	CMP	2	compare A and B and set EQ, LT or GT signals
10	CMP0	1	compare A with 0 and set EQ, LT or GT signals
11	NEGATE	1	A = NOT(A)
12	MOD	3	C = A Modulus B
13	POW	3	C = A B
14	DC2	5	decode A into B, C, D, and E
15	EC2	5	encode A, B, C, and D into E
16	MOVE	2	B = A

FEXI

The front-end expert system interface FEXI **40** is invoked by EDSIM **21** for obtaining user design constraints. FEXI starts an interactive session with the user to determine design constraints for the target ASIC such as the following: speed, die size, number of I/O pins, development time, chip count, desired yield, chip cost, package type, technology, power consumption, production volume, pin count, design style, microprocessor type, etc. Once the user has answered the necessary inquiries regarding the design constraints, FEXI **40** utilizes a knowledge base **41** and an inference engine **42** to determine an optimum design style for the hardware components of the integrated circuit from such choices as PLD (programmable logic device), gate array, standard cell, and macro cell. It then determines the technology to be used, and finally selects a hardware cell library to be used. For the software subsystem, a microprocessor is selected among various established microprocessor designs, such as Intel 8086, Motorola 68000, Intel 80286, etc. The database **60** includes software subroutine libraries for each of the standard microprocessors. Once the microprocessor type has been selected, the appropriate software subroutine library for that microprocessor type is selected. FEXI operates as a separate program which runs in parallel with EDSIM **21**. Although illustrated in FIG. **2** separately, FEXI uses the same inference engine used by BLATH **24**.

AAF

The AAF file **22** (Antecedent-Action-Form) is the input to BLATH **24**. It contains the behavioral description of the system that is to be designed. AAF file **22** contains information about storage elements which compose the data paths, memory elements and external connections to the system. This information is contained in the first section of the file. The rest of the file contains a description of the behavior of the system. The first line of the AAF file **22** contains the name of the system and is described by the following form:

name <name>

where <name> is the name of the system.

Storage elements are described at the top of the AAF file **22**. Storage elements can be data paths, external connections or memory elements. They are defined using the "data path", "data mask", "rom" and "ram" statements. These statements are described by the following forms:

"data path" <list>
"data mask" <list>
"rom" <list>
"ram" <list>

where <list> is a list of definitions. These lists are further described below.

Data Paths are defined in the "data path" statement. Data paths are described by the following form:

<pathname>'<'<startbit>':'<stopbit>'>'

<Pathname> is the name of the data path. <Startbit> and <stopbit> are the beginning and ending bit positions of the path. Data Paths must be defined before they can be used by the rest of the AAF.

External Connections behave just like data paths except that they define an external connection to the system. An external connection is described just like a data path in the "data path" statement, except that it is preceded by an "⊥" to indicate that it is an external connection.

Data mask provide a means of specifying a partial bitfield that is to be treated as another data path. Data mask are described by the following form:

<maskname>'<'<startbit>':'<stopbit>'>"='
<pathname>'<'<startbit>':'<stopbit>'>'

<Maskname> is the name of the mask that is to be treated as a data path. <Pathname> is the name of the data path that the partial bitfield comes from. <Startbit> and <stopbit> describe the bitrange of the mask and the bitfield of the data path.

Memory elements define RAMs and ROMs to be used in the system. Memories are defined in the "rom" and "ram" statements. Memory elements are described by the following form:

<memname> '[' <startaddr> ':' <stopaddr> ']' <startbit> ':' <stopbit>

where <memname> is the name of the memory. If the memory is a rom, then this name is also the name of a file containing the contents of the rom. <Startaddr> and <stopaddr> are the beginning and ending addresses of the memory. <Startbit> and <stopbit> are the beginning and ending bit positions of the word contained in the memory.

An example is given to illustrate the top part of an AAF file.

name	cpuexample;
data path	acc<0:15>, ir<0:15>, mar<0:15>, iar<0:15>, mdata<0:15>, sum<0:15>;
data mask	adr<0:9> = ir<0:9>;
rom	mem[0:65535]<0:7>;

The behavior of the system is described by state transition information and actions. The state transition information describes the control flow of the system. The actions describe what happens at each state. The representation of the state transition and action information is described in the following sub systems.

State transitions are defined as either being direct or conditional. Direct transitions are described by the following form:

<state1>:<state2>

where <state1> and <state2> are two state names. Direct transitions are an unconditional transition from state1 to state2. This transition will always occur when the system is in state1.

Conditional transitions are described by the following form:

<state1>:.<condition> <state2>

where <state1> and <state2> are the names of two states. <Condition> is the condition that must evaluate to true in order for this transition to be made. The condition is described as an AND function where signals and their compliments are ANDed. Compliments are shown by a "!" and ANDing is described by "*". Some sample conditions are shown below.

a*!b
!GT*!EQ
dog*cat*!bird

State actions consist of macros and assertions. Macros are described by the following form:

<macroname>'.'<instance>'(' <paramlist> ')'

<Macroname> is the name of the macro such as ADD or MOVE. <Instance> is a unique number for each instance of a macro. This allows distinctions to be made

between the same macro at different states. <Paramlist> is a list of zero or more parameter names separated by commas. Macros describe the actions that are to be taken when the system is at a given state. There are four types of parameters that can be used in a macro: Bus, Control, Signal, and Memory.

Bus parameters refer to the data paths or data mask that the macro is applied to. The data path or data mask must be defined in the appropriate statement at the top of the file. Optionally, a bitfield may be specified in the parameter list that will be used instead of the default of using the entire bitrange of the data path.

Control parameters indicate control signals that are necessary for the macro's actions. Controls are defined in the database definition of a macro and used by the rules.

Signal parameters indicate outputs from the macro that go to the controller to indicate the results of the macro. Signals are also defined in the database definition of a macro. Memory parameters refer to a memory that has been previously defined.

An example of the second part of an AAF is given. This example together with the previous example forms a complete AAF.

```
{
start:          ads;
ads   :         ift;
ift   :         dec;
dec   :         excstart;
excend:         end;
end   :         start;
lda   :         excend;
sta   :         excend;
add   :         excend;
bra   :         excend;
brp   :         excend;
excstart        :.      z2 lda;
excstart        :.      !z2*z3 sta;
excstart        :.      !z2*!z3*z4 add;
excstart        :.      !z2*!z3*!z4*z5*!z0 bra;
excstart        :.      !z2*!z3*!z4*z5*z0 brp;
ads     ::      MOVE.1( iar, mar );
ads     ::      INCR.1( iar );
ift     ::      STORE.1( mem, mar, mdata );
ift     ::      MOVE.2( mdata, ir );
dec     ::      MOVE.3( adr, mar );
dec     ::      DECODE.1( ir<10:15> );
lda     ::      STORE.2( mem, mar, mdata );
lda     ::      MOVE.4( mdata, acc );
sta     ::      LOAD.1( mem, mar, acc );
add     ::      STORE.3( mem, mar, mdata );
add     ::      ADD3.1( mdata, acc, sum );
add     ::      MOVE.5( sum, acc );
bra     ::      MOVE.6( ir<0:9>, iar );
brp     ::      MOVE.7( ir<0:9>, iar );
}
```

BLATH

To design a VLSI system from a flowchart description of a user application, it is necessary to match the functions in a flowchart with hardware cells or software subroutines from the hardware cell library **61** or the software subroutine library **62** (FIG. 2). This mapping preferably utilizes artificial intelligence techniques since the selection process is complicated and is done on the basis of a number of design parameters and constraints. The concept used for selection is analogous to that used in software compilation. In software compilation a number of subroutines are linked from libraries. In the design of VLSI systems, a functional macro can be mapped to members in the hardware cell library or software subroutine library. BLATH uses a rule based expert system to select the appropriate hardware cell or software subroutine to perform each action. If the hardware cell library has a number of cells with different geometries for performing the operation specified by the macro, then an appropriate cell can be selected on the basis of factors such as cell function, process technology used, time delay, power consumption, etc.

The knowledge base of BLATH contains information (rules) for:

1) selection of macros
2) merging of two macros
3) mapping of macros to cells
4) merging two cells
5) error diagnostics

The above information is stored in the knowledge base as rules. The first step of cell list generation is the transformation of the flowchart description into a block list. The block list contains a list of the functional blocks to be used in the integrated circuit. The BLATH maps the blocks to cells selected from the cell library, selecting an optimum cell for a block through the use of rules in the knowledge base. The rules which are applied at this point accomplish the following:

Map arguments to data paths
Map actions to macros
Connect these blocks

The rules used by BLATH have the following format: rule name

```
        if
                ( condition 1 )
                ( condition 2 )
                  ..
                  ..
                ( condition n )
        then
                ( action 1 )
                ( action 2 )
                  ..
                  ..
                ( action m ).
```

Exemplary of the rules used by BLATH are the following:

```
Rule 1
        IF      no blocks exist
        THEN
                generate a system controller.
Rule 2
        IF      a state exists which has a macro AND
                this macro has not been mapped to a block
        THEN
                find a corresponding macro in the library
                and generate a block for this macro.
Rule 3
        IF      there is a transition between two states AND
                there are macros in these states using the same
                argument
        THEN
                make a connection from a register corresponding to the first macro to another register
                corresponding to the second macro.
Rule 4
        IF      a register has only a single connection from
                another register
        THEN
                combine these registers into a single register.
Rule 5
        IF      there are two comparators AND
                input data widths are of the same size AND
                one input of these is same AND
                the outputs of the comparators are used to
```

-continued

		perform the same operation.
	THEN	
		combine these comparators into a single comparator.
Rule 6		
	IF	there is a data without a register
	THEN	
		allocate a register for this data.
Rule 7		
	IF	all the blocks have been interconnected AND a block has a few terminals not connected
	THEN	
		remove the block and its terminals, or issue an error message.
Rule 8		
	IF	memory is to be used, but a block has not been created for it
	THEN	
		create a memory block with data, address, read and write data and control terminals.
Rule 9		
	IF	a register has a single connection to a counter
	THEN	
		combine the register and the counter; remove the register and its terminals.
Rule 10		
	IF	there are connections to a terminal of a block from many different blocks
	THEN	
		insert a multiplexor: remove the connections to the terminals and connect them to the input of the multiplexor; connect the output of the multiplexor to the input of the block.

Additional rules address the following points:
remove cell(s) that can be replaced by using the outputs of other cell(s)
reduce multiplexor trees
use fan-out from the cells, etc.

Database

The database 60 stores information relating to hardware cells, software subroutines, macro functions, user information, and the like. A database interface 65 is provided to allow the system manager to make additions, deletions and modifications to the database. A user table 66 maintains information for every valid user of the ISSC system, including the user identification name, number and password. In the hardware cell library 61 various types of data are stored for each cell, including:

1) functional level information: description of the cell at the register transfer level
2) logic level information: description and terms of flip flops and gates
3) circuit level information: description at the transistor level
4) layout level information: geometrical mask level specifications

A cell table within the hardware cell library 61 contains a record for each cell in the cell library. Every cell in the cell table can be used as instance cell of a larger cell. Once a cell becomes an instance of a larger cell, a data dependency is created. Once this occurs, the contents in the cell can not be altered until the data dependency is eliminated. This information is kept in the attribute "times_used". The attributes of a cell as kept in the cell table are summarized in Table 2:

TABLE 2

Key	Name	Type	Integrity	Description
1	cell_id	i4	>=0	cell identification number
	cell_name	c12	alpha-numeric	cell name assigned by designer
	user_id		i4>=0	user id of the owner of cell
	width		i4>0	width of cell in centi-microns
	height		i4>0	height of cell in centi-microns
	cif_name	c20	unix file name	name of the cif file
	technology_id	i4	>=0	id of associated technology
	test_id		i4>=0	id of the test tool(s) used
	macro_id	i4	>=0	id of macro function for cell
	delay		f4>0	max. prop. delay in nano second
	power		i4>0	power consumption in micro watt
	net_name	c20	unix file name	name of the netlist file
	protection	i4	>=0	flags for user, group, world prot.
	times-used	i4	>=0	counts cell reference quantity
	date	date		date created

An example of a cell is shown below:

cell_id	23
cell_name	XIN02
user_id	8946
width	68
height	2.2
cif_name	xin02.cif
technology_id	1
test_id	1
macro_id	4
delay	0.3
power	25
net_name	xin02.mdl
protection	555
date	88-11-10

Each terminal for a cell is kept in a terminal table. Terminals can be signal, power or ground. The cell_id determines which cell the terminal belongs to. In order to uniquely identify the terminal, both cell_id and terminal_id are required. More than one terminal per cell can have the same terminal name but they are distinguished by the conjunction of cell_id and terminal_id.

The attributes of the terminal table are set forth in Table 3.

TABLE 3

Key	Name	Type	Integrity	Description
1	cell_id	i4	>=0	cell identification number
2	terminal_id	i4	>=0	terminal identification number
	name	c20	alpha-numeric	terminal name
	type	c5	alpha-numeric	terminal type (vdd, gnd, signal)
	layer	c5	alpha-numeric	terminal layer (m1, poly, diff)
	direction	c1	l,r,t,b	terminal exit side
	x	i4		x coordinate of the terminal
	y	i4		y coordinate of the terminal
	width	i4		terminal width

The cellref table contains the hierarchical cell structure. Using this table, it is possible to learn if a cell has any parent, and therefore a data dependency. The attributes of the cellref table are set forth in Table 4:

TABLE 4

Key	Name	Type	Integrity	Description
1	parent_id	i4	>=0	cell id of the parent
2	child_id	i4	>=0	cell id of the child
3	instance_id	i4	>=0	instance id to distinguish cells
	x	i4		x coordinate of the instance
	y	i4		y coordinate of the instance
	MX	i2	Boolean	True if inst. is mirrored in X
	MY	i2	Boolean	True if inst. is mirrored in Y
	RO	i2	Boolean	Rotation variable in X
	R1	i2	Boolean	Rotation variable in Y

Technology refers to a particular process run of a IC foundry. For each entry in the technology table, there is a set of cells that uses the technology (called cell library). Each record in this table is a unique process run to not only within the foundry, but also within the corporation. The attributes of the technology table are set forth in Table 5 below:

TABLE 5

Key	Name	Type	Integrity	Description
1	technology_id	i4	>=0	id that identifies technology
	facility	c20	alpha-numeric	name and location of fab house
	device	c10	alpha-numeric	CMOS, bipolar, HCMOS, GaAs, etc.
	feature_size	i4	>0	channel width of a transistor
	no_metal	i2	>=0	number of metals used
	no_poly	i2	>=0	number of polysilicons used
	well_type	c10	alpha-numeric	N-Well, P-Well, etc.

Every standard frame used for a specific technology is described in the frame table. Frame table is related to the technology table by the key technology_id. Each frame must belong to one and only one technology, while each technology can have many frames. Each record in the frame table represents a frame. The attributes of the frame table are set forth in Table 6 below:

TABLE 6

Key	Name	Type	Integrity	Description
1	frame_id	i4	>=0	id that identifies each frame
2	technology_id	i4	>=0	id of corresponding technology
	no_pads	i2	>0	number of pads in the frame
	in_width	i4	>=0	width of usable space in frame
	in_height	i4	>=0	height of usable space in frame
	out_width	i4	>0	total frame width with pads
	out_height	i4	>0	total frame height with pads

The software subroutine library 62 includes for each microprocessor type, predefined software subroutines which correspond to the macro functions which can be implemented in software. Typical subroutines used in ISSC are standard I/O macros, data conversion macros, character conversion macros, and string manipulation macros.

Every macro function used in categorizing a cell in terms of its function is stored as a single record entry in a macro table contained within the macro function library 63. Many cells within the cell library can be capable of performing the function of a macro. Each macro in the table is uniquely identified by the key macro_id. The attributes of the macro table are set forth in Table 7:

TABLE 7

Key	Name	Type	Integrity	Description
1	macro_id	i4	>=0	id that identifies each macro
	macro_name	c20	alpha-numeric	macro name describing function
	no_inputs	i2	>=0	number of inputs in function
	no_outputs	i2	>=0	number of outputs in function
	description	c80	sdl	description

CONGEN

CONGEN 27 used the STF file from BLATH 24 to design a PLA based system controller for the chip which generates control signals to enable the respective hardware cells or the microprocessor. The STF file is translated to Boolean equations for PLA logic that implements next state codes and output functions. A unique code is assigned for each state by applying heuristic rules for state assignment. The output of CONGEN is an input parameter file for automatic PLA layout generation.

SOFTCOM

SOFTCOM 70 takes as input the STF file 26 created by BLATH 24 and generates an assembly program which is the monitor program for the microprocessor. Some of the variables of this program are:

NUM_SHARED is the number of shared variables between software and hardware

REG_ARRAY[0.. NUM_SHARED] is a external register array.

NUM_SUB is the number of software routines in the system.

SHARED_VARS is an array of memory locations at which the shared variables are stored.

change_flag: associated with each shared variable is a flag which is set if that variable's value is changed.

The program format is as follows:

```
1)  set change_flag for shared variables to false.
2)  get subroutine code from REG_ARRAY[0]
3)  if code = = 1 then call subroutine 1
    if code = = 2 then call subroutine 2
    . . .
    . . .
    if code = = NUM_SUB then call subroutine_num_sub
4)  for i = 1 to NUM_SHARED do
    {
        if change_flag set then
        move SHARED_VARS[i] TO REG_ARRAY[i]
    }
5)  transfer control to the system controller
```

Design Example

This example illustrates the design approach employed in designing an application specific integrated circuit which does simple computer graphics operations such as translation, mirror image of X axis or Y axis, and rotation. All of those operations are involved in matrix multiplication with the transformation axis. The matrix multiplication will be implemented in hardware to speed up the process. FIG. 5 is a flowchart illustrating a first software portion (software portion #1) of the foregoing program which is to be processed by the microprocessor subsystem within the target ASIC. FIG. 6 is a continuation of the flowchart of FIG. 5 and illustrates the remaining portion of software portion #1 as well as software portion #2 to be processed by the microprocessor subsystem within the target ASIC, and additionally shows the non-microprocessor hardware subsystem within the target ASIC.

FIG. 7 is a block schematic diagram similar to the general diagram of FIG. 3 but showing the components which would be utilized in the above-noted specific example. To avoid repetitive description, elements in FIG. 7 which correspond to elements previously described in connection with FIG. 3 will be identified by the same reference characters with prime notation (') added. The microprocessor subsystem 80' contains CPU 81' and RAM 82' (main memory). The execution code of software portions #1 and #2 (82*a*, 82*b*) are stored in the main memory of microprocessor subsystem. The data segment 82*c* contains the data to be processed by both CPU and non-microprocessor hardware subsystem. The data in data segment 82*c* are sent to data registers via the system data bus 83'. The communication between microprocessor subsystem 80' and hardware subsystem 90' is coordinated by using interrupts. After software portion #1 (82*a*) is executed, microprocessor subsystem 80' reads the status registers to see if hardware subsystem 90' is busy. If hardware subsystem 90' is idle, its status registers are set, and the hardware subsystem 90' is activated.

When hardware subsystem 90' finishes the task, it generates an interrupt signal to microprocessor subsystem 80'. The interrupt is serviced by interrupt handler 82*d*. The main tasks of the interrupt handler include:

push contents of microprocessor subsystem 80' registers into stack.

transfer data from hardware subsystem 90' to microprocessor subsystem 80'.

reset the status register of hardware subsystem 90'.

restore the contents of microprocessor subsystem 80' registers from stack.

continue the execution of software portion #2 (82*b*).

The execution flow of the entire flowchart (including software and hardware portions) is as follows:

1. Microprocessor subsystem 80' executes the codes in software portion #(82*a*).
2. Microprocessor subsystem 80' activates hardware subsystem 80' by overwriting the status registers, then sends the data to data registers.
3. Hardware subsystem 90' processes the data in data registers
4. After hardware subsystem 90' processed the data, it sends interrupt signal to microprocessor subsystem 80'.
5. The interrupt handler 82*d* of microprocessor subsystem 80' handles the interrupt, transfers the data from data registers back to microprocessor subsystem, and resets the status registers. After the interrupt is processed, the execution flow is returned to software portion #2.
6. Microprocessor subsystem 80' executes the codes in software portion #2.
7. Done.

As shown in FIG. 6, states 24–35 and conditions 9–11 are implemented in hardware subsystem to perform multiply/accumulate operation. A multiplier and an adder are used in state 28 to implement the multiply/accumulate operations. The blocks indicated as state 24, state 25, state 27, state 29, state 32, and state 34 use counters i, j, k, which have increment and clear capabilities. In state 26 and state 31, temp is used as a temporary register to store the intermediate result. State 30, state 33, and state 35 use a comparator. The result of the comparison will affect the control flow which specified in cond9, cond10, and cond11.

The execution flow of hardware subsystem is as follows:

1. Microprocessor subsystem sends a start signal to hardware subsystem via system control bus.
2. All the data are loaded to data registers via system data bus.
3. Loop counters i, j, and k are reset to zero initially. Register temp is also reset to zero.
4. Multiply/Accumulate operation.
5. Increment the loop counters and check if the end of the loops. If it is not end of the loops, go to step 3; else exits the loops.
6. The result of multiply/accumulate operation is stored in the output register that is connected to system data bus.
7. An interrupt signal is sent to the microprocessor subsystem via system control bus.

While the invention has been described herein with reference to a specific embodiment, the description is illustrative of the invention and is not to be construed as limiting the invention. Various modifications and applications may occur to those skilled in the art without departing from the true spirt and scope of the invention as defined by the appended claims.

That which is claimed is:

1. A computer-aided design system for designing an application specific integrated circuit from architecture independent functional specifications for the integrated circuit, whose intended function is implemented by both a hardware subsystem including hardware elments on the integrated circuit and by a software subsystem including software instructions executed by a microprocessor on the integrated circuit, comprising

input means operable by a user for defining architecture independent functional specifications for the application specific integrated circuit wherein at least one of said architecture independent functional specifications is free of indication that an intended function of said integrated circuit is implanted on said hardware subsystem or said software subsystem, and

computer operated means for translating the architecture independent functional specifications into an architecture specific structural level definition of the integrated circuit, said computer operated translating means comprising

means defining software instructions of the software subsystem from the architecture independent functional specifications, at least one of which is free of indication that an intended function of said inte-

specific integrated circuit selected from the group consisting of annual volume, speed, size, pin count, packaging type, power consumption, and thermal stability.

30. A process as defined in claim 29 wherein said knowledge base additionally contains design style rules for selecting, based upon the design constraints, an optimum design style for implementation of the hardware subsystem, and wherein said step of applying the rules of the knowledge base includes selecting, based upon the design constraints, an optimum design style for implementation of the hardware subsystem.

31. A process as defined in claim 30 wherein said design style is selected from the group consisting of programmable logic device (PLD), gate array, standard cell, and macro cell.

32. A process as defined in claim 25 wherein said step of specifying a macro selected from the macro library further includes indicating, for at least certain ones of the specified macros, that the selection of whether the macro function is to be implemented in hardware or in software is to be made by the computer system, and

wherein said knowledge base additionally contains implementation rules for determining the optimum implementation of macro functions either in the hardware subsystem or in the software subsystem, and

wherein said step of applying the rules of the knowledge base includes determining, based upon the implementation rules in said knowledge base, the optimum implementation of the macro function either in the hardware subsystem or the software subsystem.

* * * * *